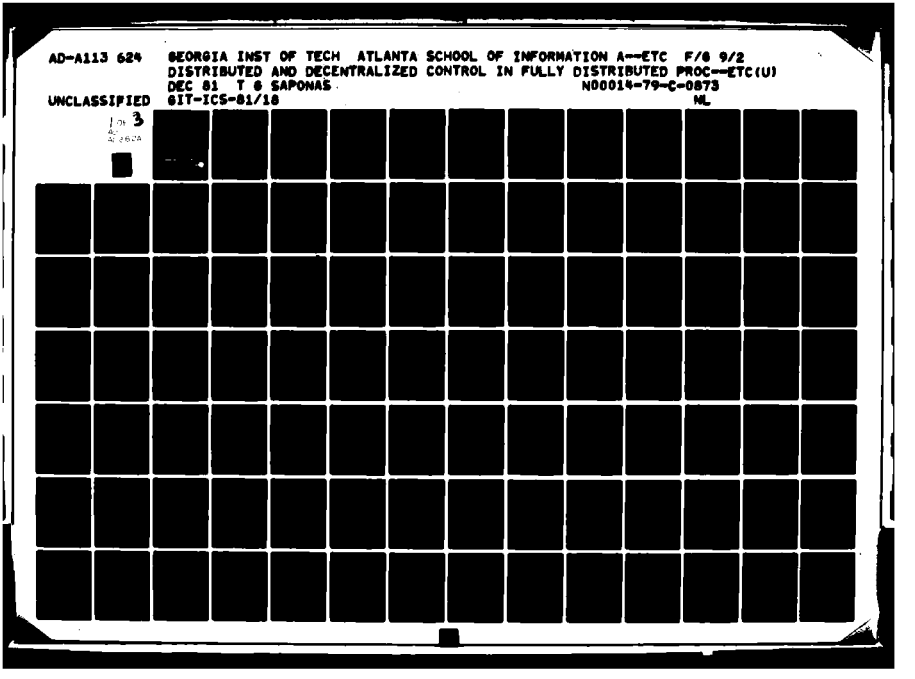

AD-A113 624
GEORGIA INST OF TECH ATLANTA SCHOOL OF INFORMATION A--ETC F/G 9/2
DISTRIBUTED AND DECENTRALIZED CONTROL IN FULLY DISTRIBUTED PROC--ETC(U)
DEC 81 T G SAPONAS
N00014-79-C-0873
UNCLASSIFIED
GIT-ICS-81/18
NL
3

(12)

AD_____

TECHNICAL REPORT
GIT–ICS–81/18

DISTRIBUTED AND DECENTRALIZED CONTROL IN FULLY DISTRIBUTED PROCESSING SYSTEMS

By

Timothy G. Saponas

Prepared for

OFFICE OF NAVAL RESEARCH
800 N. QUINCY STREET
ARLINGTON, VIRGINIA 22217

Under

Contract No. N00014–79–C–0873
GIT Project No. G36–643

ROME AIR DEVELOPMENT CENTER (ISCP)
DEPARTMENT OF THE AIR FORCE
GRIFFISS AIR FORCE BASE, NEW YORK 13441

Under

Contract No. F30602–78–C–0120
GIT Project No. 654

December 1981

GEORGIA INSTITUTE OF TECHNOLOGY
A UNIT OF THE UNIVERSITY SYSTEM OF GEORGIA
SCHOOL OF INFORMATION AND COMPUTER SCIENCE
ATLANTA, GEORGIA 30332

THE RESEARCH PROGRAM IN
FULLY DISTRIBUTED PROCESSING SYSTEMS

DISTRIBUTED AND DECENTRALIZED CONTROL

IN

FULLY DISTRIBUTED PROCESSING SYSTEMS

TECHNICAL REPORT

GIT-ICS-81/18

Timothy G. Saponas

December, 1981

Office of Naval Research
800 N. Quincy Street
Arlington, Virginia 22217

Contract Number N00014-79-C-0873
GIT Project Number G36-643

Rome Air Development Center (ISCP)
Department of the Air Force
Griffiss Air Force Base, New York 13441

Contract Number F30602-78-C-0120
GIT Project Number G36-654

The Georgia Tech Research Program in
Fully Distributed Processing Systems
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

THE VIEW, OPINIONS, AND/OR FINDINGS CONTAINED IN THIS REPORT ARE THOSE OF THE AUTHORS AND SHOULD NOT BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE AIR FORCE POSITION, POLICY, OR DECISION, UNLESS SO DESIGNATED BY OTHER DOCUMENTATION.

Georgia Institute of Technology

FDPS Control

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER GIT-ICS-81/18	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Distributed and Decentralized Control in Fully Distributed Processing Systems		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER GIT-ICS-81/18
7. AUTHOR(s) Timothy G. Saponas		8. CONTRACT OR GRANT NUMBER(s) N00014-79-C-0873
9. PERFORMING ORGANIZATION NAME AND ADDRESS School of Information and Computer Science Georgia Institute of Technology Atlanta, Georgia 30332		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research 800 N. Quincy Street Arlington, Virginia 22217		12. REPORT DATE December, 1981 13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) same as item 11		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution

17 DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

same

18 SUPPLEMENTARY NOTES

This work was partially supported by Rome Air Development Center (project engineer Thomas F. Lawrence) under Contract No. F30602-78-C-0120. The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Air Force position, policy, or decision, unless so designated by other documentation.

19 KEY WORDS (Continue on reverse side if necessary and identify by block number)

Control
Decentralized Control
Distributed Processing
Fully Distributed Processing Systems
Network
Network Operating System

20 ABSTRACT (Continue on reverse side if necessary and identify by block number)

An essential component of a Fully Distributed Processing System (FDPS) is the distributed and decentralized control. This component unifies the management of the resources of the FDPS and provides system transparency to the user. In this dissertation the problems of distributed and decentralized control are analyzed and fundamental characteristics of an FDPS executive control are identified. Several models of control have been constructed in order to demonstrate the variety of resource management strategies available to system designers and provide some insight into the relative merits of the various

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

strategies. The performance of four control models has been analyzed by means of simulation experiments.

A partitioned management strategy is utilized in the first control model. In this model a global search is enlisted in order to locate all resources required to satisfy a user request. The second model of control maintains a central directory of all resources. All requests for resources must be handled by the node possessing the central directory. The third model differs from the first model in the technique used to locate available resources. In the third model a search of the resources available at the local node is conducted before any global search. Only if all resources cannot be found locally is a global search conducted. The fourth model of control maintains identical, redundant resource directories on all nodes with access to the directories provided in a serial fashion by passing a special message called the control vector among the nodes. Modifications made to a directory by the holder of the control vector are transmitted to all other nodes.

Four groups of simulation experiments were conducted in order to study the behavior of the control models in a distributed processing environment. The first group of experiments examined the behavior of jobs accessing local files while the second group investigated the behavior of jobs remotely accessing files. The third group of experiments studied jobs not requiring file access and possessing small service times were studied. A mixed population of two different types of jobs was analyzed in the fourth group of experiments. The two types of jobs corresponded to those used in the second and third group of experiments.

In the first group of experiments the average work request response time approaced a constant value, which was similar to the value obtained with a single node simulation as the communication bandwidth increased. The results of the first two groups of experiments indicated little difference in the performance of the various models. The third group of experiments, though provided a clear distinction among values for average response time for the various models with a relative ordering from smallest to largest average work request response time as follows: model 2, model 3, model 1, model 4. As the communication bandwidth was increased the distinction between the first three models diminished, but the value for average work request response time for model 4 remained consistently higher than for the other models. Finally, in the fourth group of experiments the average work request response times for the short jobs increased as the fraction of jobs accessing remote files was increased.

ABSTRACT

An essential component of a Fully Distributed Processing System (FDPS) is the distributed and decentralized control. This component unifies the management of the resources of the FDPS and provides system transparency to the user. In this dissertation the problems of distributed and decentralized control are analyzed and fundamental characteristics of an FDPS executive control are identified. Several models of control have been constructed in order to demonstrate the variety of resource management strategies available to system designers and provide some insight into the relative merits of the various strategies. The performance of four control models has been analyzed by means of simulation experiments.

A partitioned management strategy is utilized in the first control model. In this model a global search is enlisted in order to locate all resources required to satisfy a user request. The second model of control maintains a central directory of all resources. All requests for resources must be handled by the node possessing the central directory. The third model differs from the first model in the technique used to locate available resources. In the third model a search of the resources available at the local node is conducted before any global search. Only if all resources cannot be found locally is a global search conducted. The fourth model of control maintains identical, redundant resource directories on all nodes with access to

the directories provided in a serial fashion by passing a special message called the control vector among the nodes. Modifications made to a directory by the holder of the control vector are transmitted to all other nodes.

Four groups of simulation experiments were conducted in order to study the behavior of the control models in a distributed processing environment. The first group of experiments examined the behavior of jobs accessing local files while the second group investigated the behavior of jobs remotely accessing files. The third group of experiments studied jobs not requiring file access and possessing small service times were studied. A mixed population of two different types of jobs was analyzed in the fourth group of experiments. The two types of jobs corresponded to those used in the second and third group of experiments.

In the first group of experiments the average work request response times approached a constant value, which was similar to the value obtained with a single node simulation as the communication bandwidth increased. The results of the first two groups of experiments indicated little difference in the performance of the various models. The third group of experiments, though, provided a clear distinction among values for average response time for the various models with a relative ordering from smallest to largest average work request response time as follows: model 2, model 3, model 1, model 4. As the communication bandwidth was increased the distinction between the first three

models diminished, but the value for average work request response time for model 4 remained consistently higher than for the other models. Finally, in the fourth group of experiments the average work request response times for the short jobs increased as the fraction of jobs accessing remote files was increased.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SECTION 1

INTRODUCTION

Technological advances in communications have made feasible the interconnection of multiple computers and created the problem of managing the numerous resources provided by the individual systems so as to make them accessible to all users regardless of their point of entry into the distributed system. Solutions to the control problem for uniprocessors are not directly applicable to distributed processing systems due to the distributed nature of the resources. Thus, it is necessary that new resource management strategies, hereafter referred to as control strategies, be designed for distributed processing systems.

A number of distributed processing systems have been constructed each using a different control strategy (see Chapter II), but no comprehensive study of the control problem has been undertaken. This dissertation analyzes the problem of process control in a distributed processing system. Fundamental characteristics and functional requirements of the control are identified, and, from these, a number of models of control are developed to help visualize the variety of control strategies available to system designers. Finally, the performances of the various control models are analyzed by means of simulation experiments, and the models are evaluated on the basis of the performance results as well as certain qualitative features.

This dissertation is concerned with a particular class of distributed processing systems, "Fully Distributed Processing Systems (FDPS)." For a system to be classified as an "FDPS," it must possess all five of the following characteristics:

1. Multiplicity of resources
2. Component interconnection with two-party, cooperative protocols
3. Unity of control
4. System transparency
5. Component autonomy

The first characteristic requires that an FDPS be composed of a multiplicity of "general-purpose" resources. They must all be freely assigned on a short-term basis to various system tasks as required (hardware and software

processors, shared data bases, etc.). The second characteristic is that the active components in the FDPS must be physically connected by a communication network(s) utilizing two-party, cooperative protocols to control the physical transfer of data (i.e., loose physical coupling).

The FDPS must also possess an executive control that defines and supports a unified set of policies governing the operation and utilization of all physical and logical resources. In addition, the executive control must provide system transparency. Users must be able to request services by generic names without being aware of their physical location or the fact that multiple copies of the resources may exist. (System transparency is designed to aid rather than inhibit and, therefore, can be overridden. A user who is concerned about the performance of a particular application can provide system-specific information to aid in the management control decisions.)

Finally, both the logical and physical components of an FDPS should interact in a manner described as "cooperative autonomy." [Ensl78] This means that the components operate in an autonomous fashion requiring cooperation among processes for the exchange of information as well as for the provision of services. In a control environment observing the rules of cooperative autonomy, the components reserve the ability to refuse requests for service, regardless of whether the service request involves execution of a process or the use of a file. This could result in anarchy except for the fact that all components adhere to a common set of system utilization and management policies expressed by the philosophy of the executive control.

The primary task of the FDPS control is the management of system resources. This includes both physical resources (e.g., processors, memory, disks, tape drives, and printers) and logical resources (e.g., processes and files). Most methods of control currently utilized in uniprocessors and multiprocessors are inherently centralized and are based on the premise that all processes share a coherent and deterministic view of the entire system state [Jens78]. Many researchers (see for example [Ensl78, Jens78, LeLa79]) argue that a distributed and decentralized approach to control will be necessary in order to realize the advantages (e.g., extensibility, integrity, and performance) that are potentially available with the distribution of multiple resources.

"Distributed control" is characterized by having its executing components physically located on different nodes. This means there are multiple loci of control activity. "Decentralized control" means that control decisions are made independently by separately executing control components. In other words, there are multiple loci of control decision making. Therefore, a distributed and decentralized control has active components located on different nodes, and those components are capable of making independent control decisions.

The problem of control within an FDPS has been the subject of three papers [Sapo80], [Ensl81a], and [Ensl81b]. In [Sapo80] a specific model of control is described. [Ensl81a] contains an analysis of the FDPS control problem including the identification of design alternatives for an FDPS executive control and the specification of several models of control. In [Ensl81b] the models of control described in [Ensl81a] are further refined, and an analysis of the relative performance of the models is conducted using simulation techniques.

The purpose of this dissertation is to conduct a detailed analysis of the problem of controlling an FDPS with special emphasis given to distributed and decentralized techniques. In Chapter II, the control strategies used by other researchers in their distributed processing systems are examined to provide an appreciation for the variety of control strategies available to system designers. The fundamental characteristics of FDPS control are presented in Chapter III. Utilizing the design alternatives presented in Chapter III, several models of control are constructed and described in Chapter IV. In Chapter V, the method of performance analysis utilized in this work (i.e., simulation) is explained. This includes a description of both the simulator and the basic environment applicable to each of the simulation experiments. A description of each simulation experiment along with a presentation of the results is provided in Chapter VI. The simulation results are analyzed with the aid of analytical models in Chapter VII. In chapter VIII, the control models described in Chapter IV are evaluated on the basis of their performance (as demonstrated via the simulation experiments) and various qualitatively evaluated features. Finally, conclusions, recommendations, and a discussion of possible future research are presented in Chapter IX.

SECTION 2

BACKGROUND

Distributed processing systems have been in existence since the late 1950's, when the National Bureau of Standards developed the PILOT system [Lein58]. With few exceptions, nearly all systems developed until the late 1970's were either uniprocessors or tightly-coupled multiprocessors. Control in both of these types of systems is made possible through the use of highly centralized techniques based on the premise that all processes share a coherent and deterministic view of the entire system state [Jens78]. The consistency of this view and the resulting control activities is enforced by a unique, lower level entity. Examples of such low-level entities are monitors [Hoar74] and memory access control hardware.

Two examples of multiprocessor systems are the C.mmp [Wulf72, Wulf81] and the Cm* [Swan76a,b] systems, both of which were developed at Carnegie-Mellon University. C.mmp consists of a number of processors each possessing a local memory. All processors are connected to a common memory. The operating system for C.mmp consists of the kernel called HYDRA, which provides a set of mechanisms for building an operating system, and a standard extension, which implements a set of standard operating system functions (e.g., scheduler and file system). The information needed to conduct standard operating system functions is maintained in shared tables.

Cm* consists of a number of processors each possessing a local memory, but Cm* does not possess a common memory to which all processors are directly connected; instead, each processor possesses the capability of directly addressing the local memory of all other processors. This is achieved through special switches called Kmap's. Processors are collected into clusters with all processors of a cluster connected to a single Kmap. The Kmap's are interconnected in order to provide access between clusters. The Kmap is a very intelligent switch which determines if an addressed entity resides within its cluster or exists in another.

Two experimental operating systems have been developed for Cm*, StarOS [Jone79a] and Medusa [Oust80a,b]. Both of these operating systems utilize control strategies that involve the partitioning of resources and activities.

This partitioning is static and occurs during system initialization. Each operating system is constructed as a "task force" [Jone79b]. A task force consists of a group of processes cooperating and communicating to achieve a goal. Centralized tables are utilized to hold control information used by the processes of the operating system's task force.

A number of loosely-coupled distributed processing systems have been proposed, and several have actually been implemented. For most of the systems, the control strategy that is utilized falls into one of the following four categories:

1. Autocracy
2. Sequential
3. Hierarchical
4. Partitioned

An autocracy contains a single entity that unilaterally formulates and executes all decisions on all resources. With the sequential strategy, all activities are performed by one manager for a period of time and then by the next manager in succession. Another strategy is to establish management in a hierarchical manner in which managers at a given level supervise a set of managers at the next lower level. The top level may possibly contain more than one manager. Finally, there is the partitioned control strategy, in which resources are partitioned and separate managers are assigned to each partition.

There are a number of proposals for systems that utilize an autocratic form of control. The KOCOS system [Aiso75] is composed of a number of processor-memory pairs connected to a common bus via bus interface units. Control of system resources is centralized in the system scheduler which is present on only one of the processors. Control of a dynamic process is given to the local operating system that resides on the processor in which that process resides. A local operating system resides on every processor except the one containing the system scheduler.

A similar proposal for managing resources in a distributed processing system has been made by Lunn [Lunn81]. This system contains a "local available resource directory" (LARD) and a "total active resource directory" (TARD). A LARD is located on each node of a system. It contains the resour-

ces currently available at the node. All active LARDs maintain between themselves the TARD which contains information concerning all resources in the system. The manager of the TARD resides at a single node. To locate a resource, a process issues a request to its LARD which searches locally. If the resource is not found locally, the LARD forwards the request on to the TARD. Therefore, all nonlocal references will be resolved by a single centrally located component, the TARD.

A slightly different approach which can still be classified as an autocracy is the Cambridge Ring [Wilk80]. This system is composed of a number of processor-memory pairs connected in a ring. For each class of resources, a single manager (called a server) is assigned. Each server has exclusive use of a processor and must provide management services for all resources of a given type that are a part of the system. Examples of the servers include the file server (provides file management), name server (maps names to network addresses), printing server (provides access to printers), and time server (supplies the current date and time).

Another basic control strategy that has been proposed for some distributed processing systems is the "sequential approach." An example of a system utilizing this approach is the ARAMIS Distributed Computer [Caba79a,b]. The nodes of this system may be physically interconnected in any manner, but they are logically connected in a loop. Multiple, redundant copies of management information for sharable resources are maintained on each node. In addition, there is a manager on each node which provides access to sharable resources for the users attached to that node. The managers operate in a serial fashion in order to prevent access conflicts to the redundant management information. A special message called the control vector (CV) circulates around the virtual loop to control the serial operation of the managers. The node which holds the CV is permitted to update its local copy of the management information (i.e., allocate and deallocate resources). The updates made by the manager are packaged in a message called the update vector (UPV), which is passed around the loop allowing the other managers to bring their copies of the management information into a consistent state. Once the UPV returns to the manager that originally created it, the CV is sent to the next node.

The Delta distributed transaction processing system [LeLa81] utilizes a similar scheme for providing concurrency control. A control token circulates on a virtual ring carrying a sequencer which delivers sequential and unique integer values called tickets. The tickets are utilized to timestamp transactions. Once tickets have been selected by the manager at a particular node, the control token is transmitted to the successor node.

A hierarchical control strategy has been proposed in a number of systems including the Stony Brook Multicomputer [Kieb81, Muke79]. Three types of nodes compose this system, G-nodes, T-nodes, and P-nodes. The G-node is the root node. It supports a global file system and manages mass storage devices for the entire system. Each T-node supports an individual transaction file system serving the P-nodes to which it is connected. User applications are run at the P-nodes which are organized in a tree with strict superior-subordinate relationships. A superior P-node processor can preempt the activity of one of its subordinate nodes. A user interface program running on the G-processor assigns tasks to the root P-processor. This processor can assign the tasks to its subordinates who can do the same. Thus, a hierarchy of control is established.

The X-tree system [Mill81] consists of a network of nodes organized in a tree topology. Devices are attached only to the leaf nodes. Objects (e.g., data, programs, processes, directories, files, and ports) are the basic addressable units in X-tree. All objects, with the exception of ports and processes, reside only at the leaf nodes. An object's address consists of a global node address (the address of the node on which the object resides) and a local node address (the address of the object within the node).

The X-tree Operating System (XOS) is composed of five major modules: 1) the microcoded kernel, 2) the capability manager, 3) the object manager, 4) the directory system, and 5) the command interpreter. Every process can potentially access any object in the system regardless of its location because XOS provides a consistent and equivalent view of the address space to all nodes. Access to objects is controlled by the object manager. Object managers residing at leaf nodes provide access to and management of the objects resident at that node. Non-leaf object managers simply act as agents by forwarding requests for objects to the appropriate leaf nodes. The

implementation of the object managers appears to be one of the few functions in which the implementation is a direct consequence of the tree topology of the network. Most other functions appear to be implemented with identical copies resident at each node.

Another system based upon a hierarchical organization is the MICRONET system [Witt79, Witt80]. MICRONET is a packet switched network of loosely-coupled LSI-11's which are interconnected by 0.5 Mbyte/sec shared communication busses. Each computer module can access two of the many busses which are passive and function with decentralized control much like Ethernet [Metc76]. Nodes consist of a host and a communication computer.

MICROS, the operating system for MICRONET, utilizes a hierarchical control strategy. The nodes of the highest level of the hierarchy form the oligarchy; the nodes which make the middle levels are called managers; and the nodes of the lowest level are called workers. No single node controls the network; instead, the highest level of management is composed of a global control oligarchy consisting of several nodes. The members of the oligarchy exchange summary information with each other in order to preserve information in the event of a hardware failure. Subordinate nodes provide summary information to their immediate supervisors. This information includes a list of their immediate subordinates. Thus, if a node is lost, its supervisor can replace that node with one of the lost node's subordinates and as a result preserve the hierarchical structure of the network. The lowest level of the hierarchy consists of nodes called workers. These nodes support user tasks and I/O handlers.

User programming on MICROS is accomplished with the use of task forces [Jone79b]. A task force consists of a collection of cooperating tasks. The technique used to schedule task forces is called wave scheduling [vanT81]. Each middle level manager maintains an approximate count of the number of its subordinate workers which are available. The count is approximate because information concerning processor allocations or deallocations requires a certain delay in order to filter up to the appropriate superior managers. If a request for a task force of size S (i.e., the task force requires S processors) is received by a manager incapable of providing that number of processors, the task force descriptor (a structure describing the task force

requirements) is passed up the hierarchy until a suitable manager is discovered.

The manager for a task force, the task force master (TFM), maintains information concerning the availability of workers in the TFM's subtree. The TFM computes $R \geq S$, which is the number of workers it will attempt to reserve. The request for R workers is divided among the subordinate managers of the TFM. This procedure continues down the hierarchy appearing as a wave of subrequests. Hardware failures and deadlock are handled through the use of time-outs at each level of the hierarchy.

A fourth strategy for control, partitioned control, can be observed in a number of systems. This strategy involves the partitioning of resources and the assignment of separate managers to each partition. There are a number of systems that partition resources, assign managers to each partition, and rely upon communication among the managers in order to make the resources globally accessible. An example of such a system is the Advanced Distributed Application Programming Tools System (ADAPT) [Peeb80]. In this system, identical copies of a kernel are maintained at each node. The kernel is composed of several processes each performing a specific role. When a kernel process is unable to satisfy a request locally, its distant counterparts are contacted in order to solve the problem.

A similar resource management strategy is utilized in the Roscoe Distributed Operating System [Solo79]. Roscoe is designed for a network of microprocessors. All processors are identical and execute the same operating system kernel. Resource managers reside on all processors and are connected by a network of links. A Roscoe link is patterned after the concept of a link in DEMOS [Bask77]. It is a one way logical connection between two processes. It combines the concepts of a communication path and a capability. If a request for process creation cannot be handled by a resource manager at a particular node, it is sent on to another resource manager which must determine whether it should service the request or pass it along to the next resource manager.

A slightly different scheme of resource management involves bidding instead of simply passing a request from node to node searching for a node to service the request. This strategy is observed in the Distributed Computer

System (DCS) [Farb72a,b, Rowe73] and a distributed problem solver called CNET [Smit79, Smit80]. In DCS, the nodes are organized in a ring. Requests are placed on the ring, and each node is given the opportunity to bid on requests that it can satisfy. The requester chooses one of the bids after waiting a certain length of time for the bids to arrive. The requester notifies the bidder that the bid has been accepted, and both processes notify a third process, the notary, which records the contract in a central file used to limit resource allocation. The central file is used to store rough limits, which need not be accurate, and thus the central file is not considered a critical component. Using the central file, the notary decides whether or not it will ratify the contract. Once ratified, the resource allocator on the chosen node creates the desired process and returns the process name to the original requester.

The procedure for satisfying requests in CNET is referred to as the Contract Net Protocol. When a node requests that a task be performed, a task announcement message is prepared. The creator of this message is called the manager of the task. The message can be transmitted using one of the following three techniques depending upon the knowledge the task manager possesses concerning the availability of resources: 1) general broadcast, 2) limited broadcast, and 3) point-to-point. Nodes listening to the message can return bids which are subsequently evaluated by the task manager. The chosen bidder, called a contractor, is sent an award message. If no bids are received within a particular time interval, the contract message is reannounced. The task manager can terminate a contract at any time. If a node becomes idle, it can issue a node availability message.

Examples of systems utilizing four basic control strategies have been presented in this chapter. The four strategies are autocracy, sequential, hierarchical, and partitioned. This discussion should give the reader an appreciation for the variety of approaches that can be taken when choosing a strategy for the management of a system's resources. In the following chapters, a detailed study is undertaken to identify key characteristics of the control strategies for Fully Distributed Processing Systems. Several models of control will be described, and the performance of these models will be analyzed by means of simulation.

SECTION 3

FUNDAMENTAL CHARACTERISTICS OF FDPS CONTROL

In order to identify the fundamental characteristics of the executive control for a Fully Distributed Processing System, the nature of an FDPS and the applications to be executed on the FDPS must first be identified. Once this has been done, it is necessary to analyze the work that must be accomplished in order to service a given application. With this accomplished, the design alternatives for the executive control can be identified.

3.1 The Nature of an FDPS

In the first chapter, Fully Distributed Processing Systems were defined. A key point in that definition that has a large impact on the design of an FDPS executive control is that the nodes of the system are loosely-coupled. This means there is no sharable memory such as is found in C.mmp. In addition, processes executing on one node cannot directly address the memory of another node as is the case in Cm*. The result is that the executive control cannot be designed on the basis of shared tables which are accessible to components residing on multiple nodes of the system. (This is the technique used in the StarOS and Medusa operating systems for Cm*.)

The FDPS executive control must integrate and unify the physical and logical resources of the system. Users accessing the FDPS at any node must be given the potential to utilize resources on any other node in the system as well as those at the local node. Therefore, the user accesses the system as a whole rather than just one node of the network.

Access to resources must be provided in a transparent manner. Users request services and are given the resources necessary to provide the services rather than directly requesting the resources. Therefore, users need not be knowledgeable of the configuration of resources in the FDPS. It is the responsibility of the executive control to locate and acquire the necessary resources.

3.2 The Nature of User Work Requests

The traditional method for programming user applications is by means of a single monolithic program. It has been discovered, however, that many

applications are easier to implement and debug as a series of communicating tasks rather than as a monolithic program [Live78b]. Denning [Denn78] claims that this type of programming is a natural way of expressing the concurrency inherent in an algorithm. Therefore, one may expect an increase in performance by exploiting the parallelism present in the algorithm.

A number of systems support this type of programming including Mininet [Live80, Mann77], StarOS [Jone79a], Medusa [Oust80a,b], MICROS [Witt80], and TRIX [Ward80]. Mininet is a system oriented towards transaction processing on distributed data bases that exhibit locality of reference. The processing of a transaction is represented by a directed graph in which the nodes represent processes and the edges messages. StarOS, Medusa, and MICROS support task forces [Jone79b] consisting of a collection of communicating processes. This programming technique is also utilized in the TRIX system. TRIX itself can be viewed as a directed graph in which the nodes represent processes and the edges represent the communication between processes.

In this study, it is assumed that users program applications by means of communicating tasks. The collection of tasks will be referred to as a task set and can be viewed as a directed graph in which the nodes of the graph represent the tasks and the edges represent the communication between the tasks.

Users present their requests for service to the system by means of work requests programmed in a command language such as that depicted in Figure 1. (Figure 1 contains the BNF description of the command language supported by the Advanced Command Interpreter available on the Georgia Tech Software Tools Subsystem. In this dissertation, examples of work requests are presented utilizing this command language.) In the work request, a user specifies a number of tasks and the connectivity (interprocess communication) of those tasks. The work request can be viewed as a specification of a directed graph. The executive control's internal representation of a work request will be referred to as a task graph. The nodes represent tasks and the edges represent communication paths between tasks.

A node specification includes the following information: 1) an optional label to identify the node, 2) a command name which names a file that contains either executable code (object file) or other work requests (command file),

```
<work request> ::= <logical net>

<logical net> ::= <logical node> { <node separator>
                  { <node separator> } <logical node> }

<node separator> ::= , | <pipe connection>

<pipe connection> ::= [ <port> ] '|' [ <logical node number> ]
                      [ .<port> ]

<port> ::= <integer>

<logical node number> ::= <integer> | $ | <label>

<logical node> ::= [ :<label> ] [ <simple node> |
                   <compound node> ] |
                   ( <simple node> | <compound node> )

<simple node> ::= { <i/o redirector> } <command name>
                  { <i/o redirector> | <argument> }

<compound node> ::= { <i/o redirector> } '{' <logical net>
                    { <net separator> <logical net> } '}'
                    { <i/o redirector> }

<i/o redirector> ::= <file name> '>' [ <port> ]  |
                     [ <port> ] '>' <file name>  |
                     [ <port> ] '>>' <file name> |
                                '>>' [ <port> ]

<net separator> ::= ;

<command name> ::= <command file name> | <object file name>

<label> ::= <identifier>

<file name> ::= <data file name>

<identifier> ::= <letter> { <letter> | <digit> }

<integer> ::= <digit> { <digit> }
```

Figure 1. BNF for the Advanced Command Interpreter's Command Language [Akin80]

and 3) optional input/output redirection instructions. A node can be identified either by its label, if it has one, or by its position on the com-

mand line. For example, in the command below, the second node has the label 'a' and the command name 'cmnd2'.

```
cmnd1 | :a cmnd2
```

This node can be identified either by the label 'a' or its position '2' but not by its name, 'cmnd2'.

Input/output redirection is used to connect the ports of a task to files in the file system. (The default for input/output is "standard input/output;" i.e., the user's terminal.) In the example below, input port number three is connected to file 'in' and output port number one is connected to file 'out'.

```
in>3 cmnd 1>out
```

The specification of the port number in the input/output redirector is optional. If it is omitted, the next unused port number is assumed. Therefore, in the example below, output port number one is connected to file 'out1', output port number two is connected to file 'out2', and output port number three is connected to file 'out3'.

```
cmnd >out1 2>out2 >out3
```

Nodes are separated by node separators which can be either the comma symbol or the vertical bar symbol. The comma symbol is used to separate a node that does not have any of its output ports connected to other nodes. The vertical bar symbol or pipe symbol is used to identify the connection of an output port of the node immediately preceding the pipe symbol and an input port of another node. The port numbers and logical node number of the pipe specification may be omitted and default values assumed. If a port number is omitted, the next unused port number for the node possessing the port is used. The logical node number of the pipe specification identifies a node of the logical network and may either be an integer identifying the position of the node on the command line, the symbol '$' which identifies the last node on the command line, or a node label. If no other node is specified, the node immediately following the pipe symbol is assumed to be the destination of the output of the pipe.

An example of a work request utilizing this syntax is shown in Figure 2. This command consists of seven logical nodes connected in the manner depicted in the figure. It demonstrates several forms of pipe specifications including the use of labels in identifying nodes. This figure also contains a graphical

representation of the work request.

3.3 Approaches to Implementing FDPS Executive Control

There are two basically different approaches available for implementing an operating system for a distributed processing system, the base-level approach and the meta-system approach [Thom78]. The base-level approach replaces all existing software up to some interface. This may include the replacement of all operating system software and the retention of utility programs and compilers. Therefore, it is possible that with this approach software for local control functions such as memory management and process management will need to be developed. In contrast, the meta-system approach utilizes the "existing" operating systems, called local operating systems (LOS), already operating on each of the nodes of the system. Each LOS is "interfaced" to the distributed system by a network operating system (NOS) which is designed to provide high level services available on a system-wide basis. The most common reason for taking the meta-system approach is the availability of existing software for accomplishing local management functions, thus providing the opportunity for reducing development costs [Thom78].

Figure 3 depicts a logical model applicable to an FDPS executive control utilizing either approach. The LOS handles the low-level (processor-specific) operations required to interface directly with users and resources. In the meta-system approach, the LOS represents primarily the operating systems presently available. The LOS resulting from a base-level approach has similar functionality; however, it represents a new design, and certain features may be modified in order to allow the NOS to provide certain functions normally provided by the LOS. Any "network" operations are performed by the NOS. System unification is realized through the interaction of NOS components, possibly residing on different processors, acting in cooperation with appropriate LOS components. Communication among the components is provided by the message handler which utilizes the message transport services which actually move the messages.

3.4 Information Requirements

The two types of data required by an executive control are information concerning the structure of the set of tasks required to satisfy the work

Work Request:

```
pgm1 | pgm2 1|a 2|b :a pgm3 | pgm4 |c.1 :b pgm5 | pgm6 |.2 :c pgm7
   (0)     (1) (2) (3)    (4)    (5)   (6)    (7)    (8)  (9)
```

(0) Output port 1 of pgm1 is connected to input port 1 of pgm2.
(1) Output port 1 of pgm2 is connected to input port 1 of the logical node labeled "a," pgm3.
(2) Output port 2 of pgm2 is connected to input port 1 of the logical node labeled "b," pgm5.
(3) Label for the logical node containing pgm3 as its execution module.
(4) Output port 1 of pgm3 is connected to input port 1 of pgm4.
(5) Output port 1 of pgm4 is connected to input port 1 of the logical node labeled "c," pgm7.
(6) Label for the logical node containing pgm5 as its execution module.
(7) Output port 1 of pgm5 is connected to input port 1 of pgm6.
(8) Output port 1 of pgm6 is connected to input port 2 of pgm7.
(9) Label for the logical node containing pgm7 as its execution module.

Data Flow Graph of the Work Request:

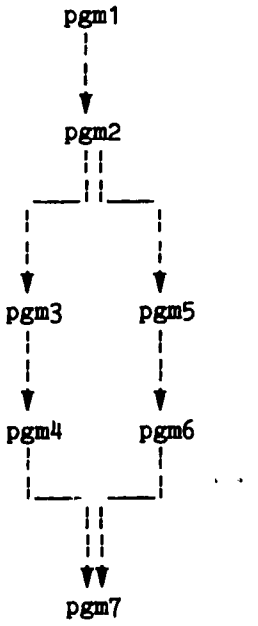


Figure 2. Example of a Work Request

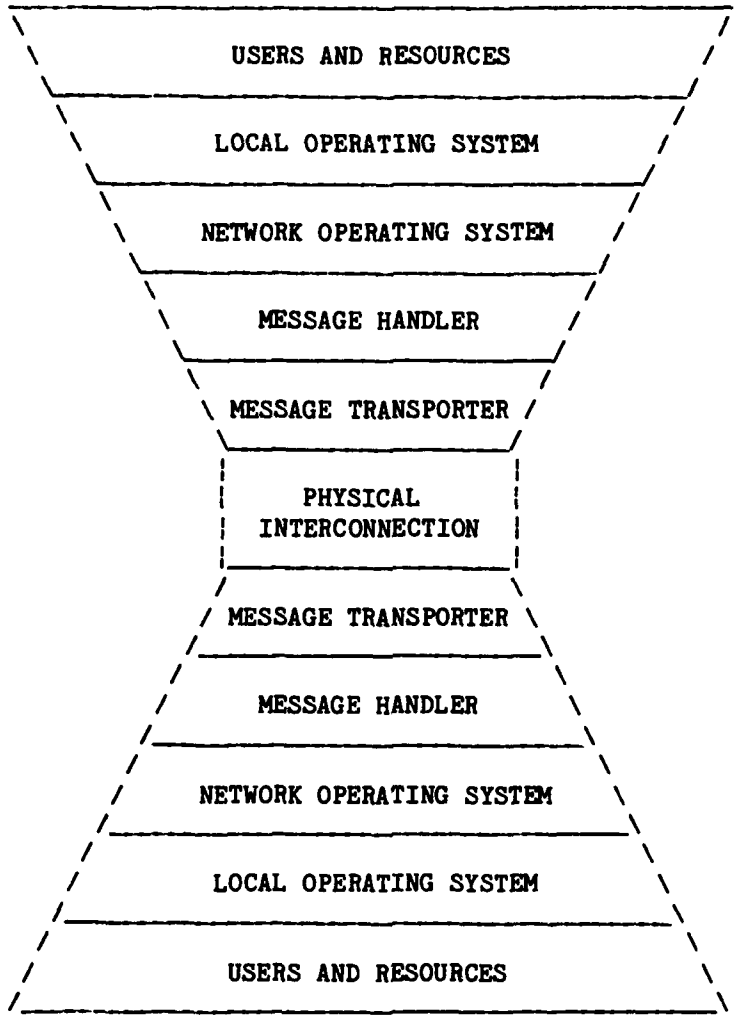


Figure 3. A Logical Model of an FDPS

request and information about system resources. This data is maintained in a variety of data structures by a number of different components.

3.4.1 Information Requirements for Work Requests

Each work request identifies a set of cooperating tasks and the connectivity of these tasks. Work requests as linear textual forms can be easily accepted and manipulated by the computer system; however, task graphs, which

are the internal control structures used to describe work requests, must be represented in a manner such that the linkage information is readily available. Two possible methods for representing the task graph are the following: 1) a linked list of node control blocks (Figure 4), or 2) an interconnection matrix (Figure 5).

Information concerning a particular task is maintained in a node control block (Figure 4). Associated with each logical node is an execution file, a series of input files, and a series of output files. The node control block contains information on each of these resources including the name of the resource, the locations of possible candidates that might provide the desired resource, and the location of the candidate resource chosen to be utilized in the satisfaction of the work request. In addition to this information, the node control block maintains a description of all interprocess communication (IPC) in which the node is a party. This consists of a list of input ports and output ports. (Interprocess communication is a term describing the exchange of messages between cooperating processes of a work request.) Typically, a message is "sent" when it is written to the output port of a process. The message is then available for consumption by any process possessing an input port that is connected to the previously mentioned output port. The message is actually consumed or accepted when the process owning the connected input port executes a READ on that port.

A global view of interprocess communication is provided by the node interconnection matrix (Figure 5). This structure indicates the presence or absence of an IPC link between an output port of one node and an input port of another node. Thus, links are assumed to carry data in only a single direction.

An example of a task graph resulting from the work request in Figure 2 utilizing the direct linking of node control blocks is presented in Figure 6. Figure 7 illustrates the utilization of an interconnection matrix.

3.4.2 Information Requirements for System Resources

Regardless of how the executive control is realized (i.e., how the components of the executive control are distributed and how the control decisions are decentralized), information concerning all system resources (processors, communication lines, files, and peripheral devices) must be maintained. This

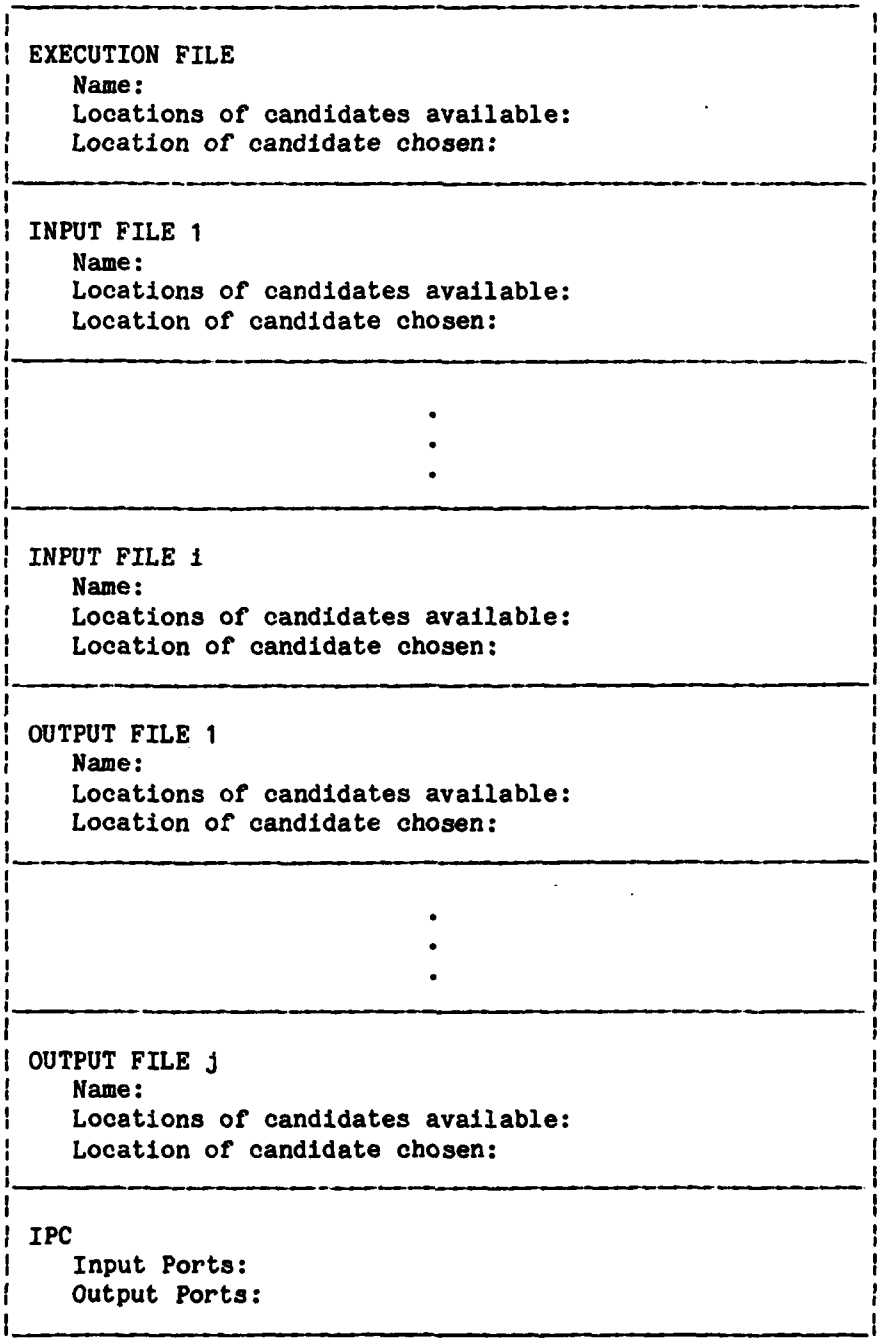


Figure 4. Node Control Block

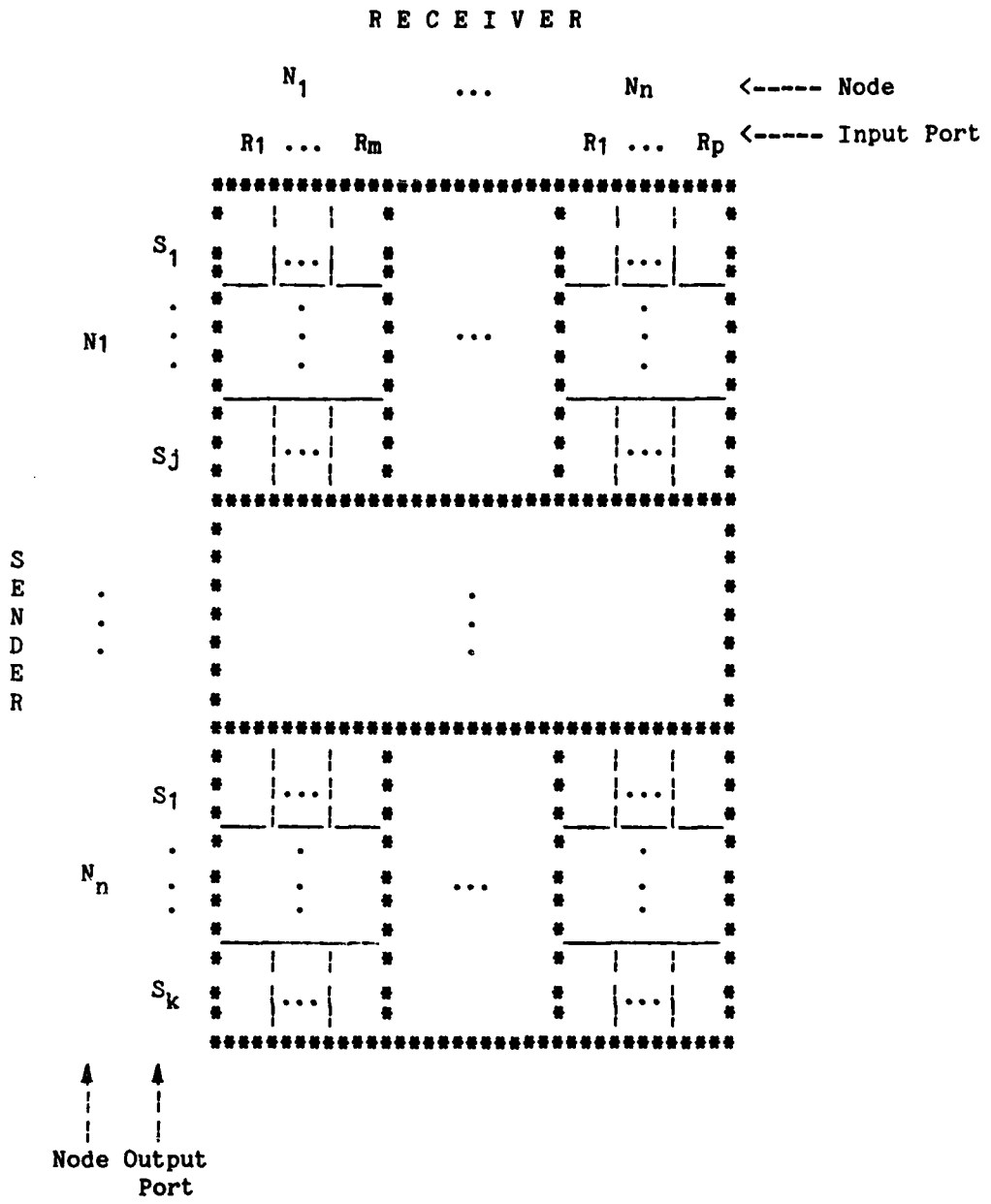


Figure 5. Node Interconnection Matrix

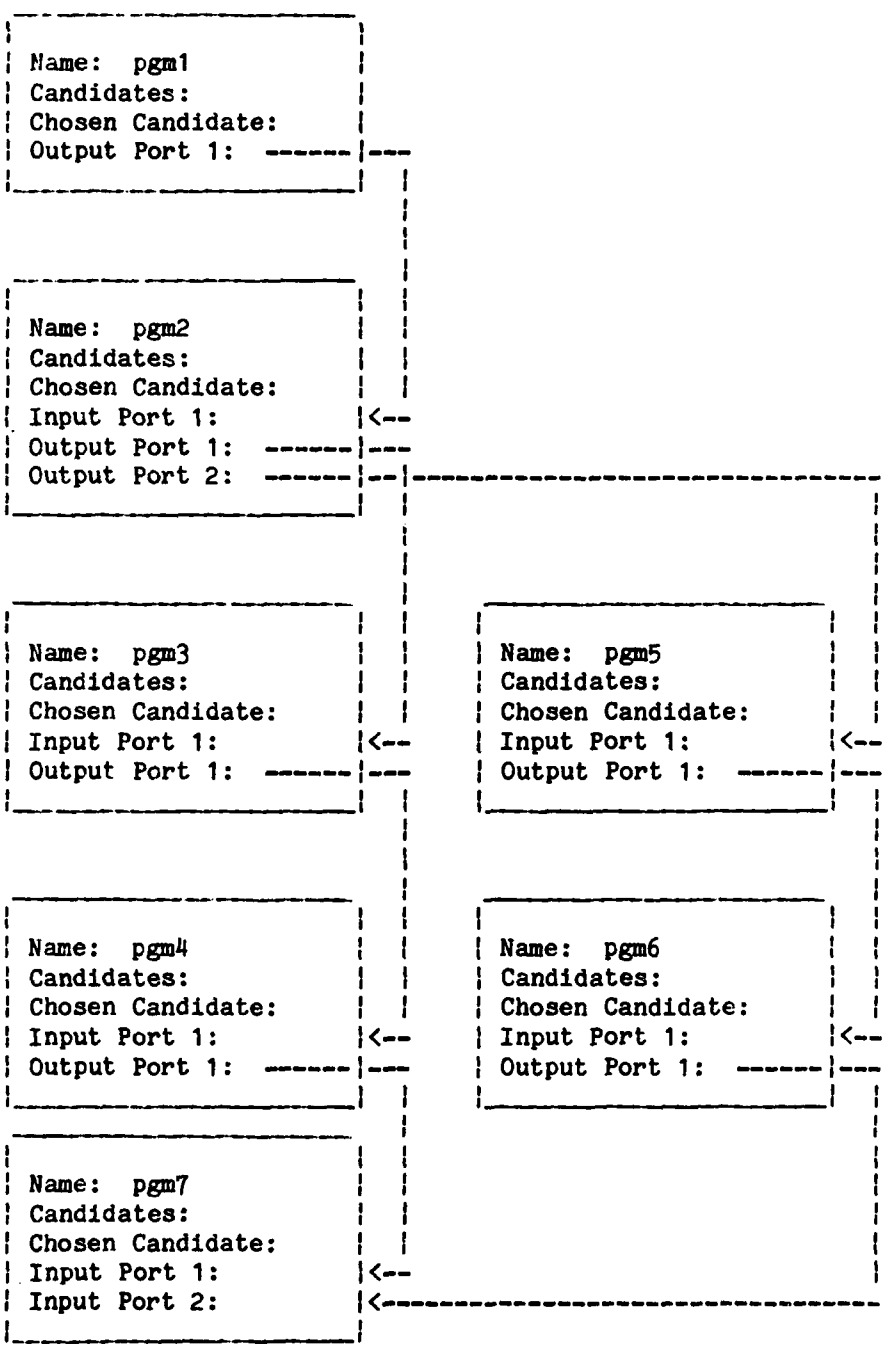


Figure 6. Example of a Task Graph Using Links within the Node Control Blocks

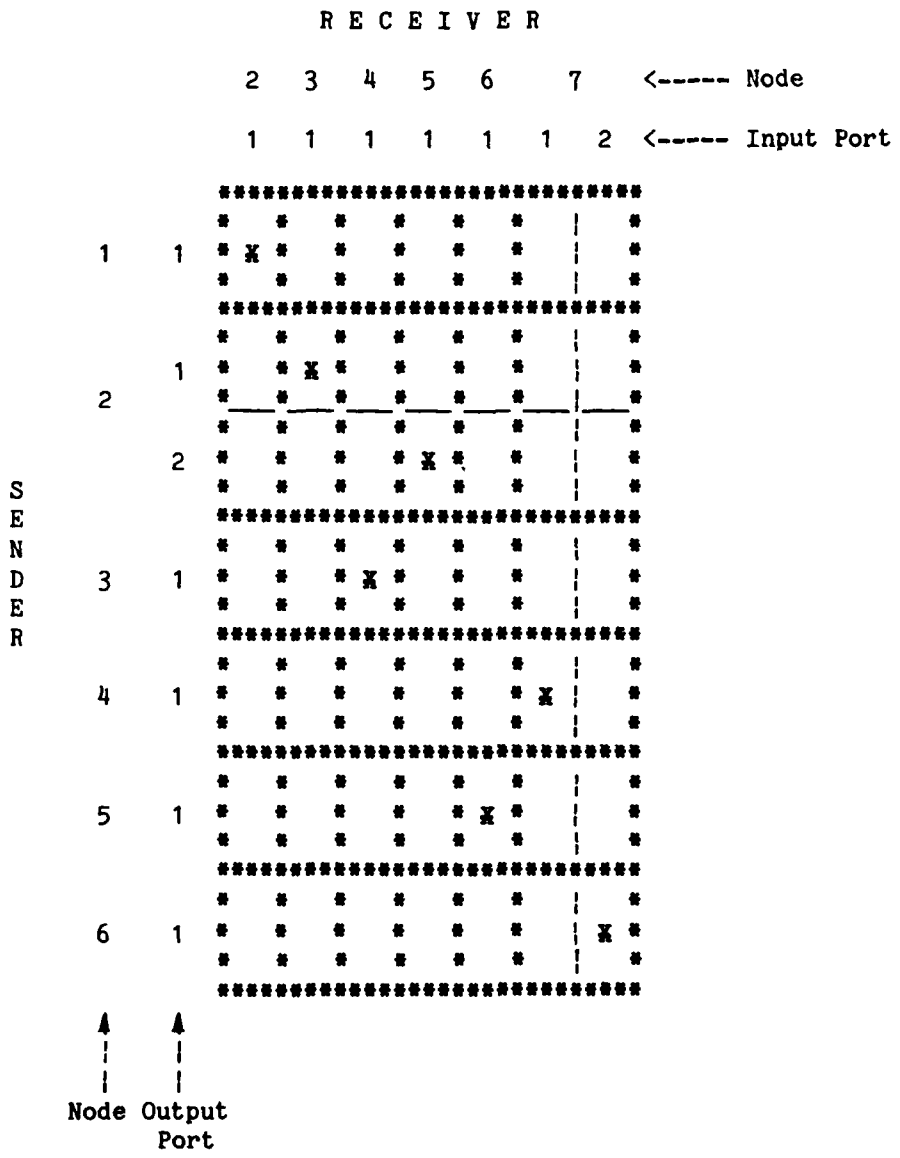


Figure 7. Example of a Node Interconnection Matrix

information includes, as a minimum, an indication of the availability of resources (available, reserved, or assigned). Preemptable resources (e.g., processors and communication lines) capable of accommodating more than one user at a time may also have associated with them utilization information designed to guide an executive control in its effort to perform load balancing.

6.5 Basic Operations of FDPS Control

The primary task of an executive control is to process work requests. A work request can be represented by a directed graph called a task graph. A node of a task graph specifies an execution file and multiple input and output files. The execution file may contain either object code or commands (work requests). All three types of files may reside on one or more physical nodes of the system, for there may be multiple copies of the same file available. Thus, to process a work request, an FDPS executive control must perform three basic operations: 1) gather information, 2) distribute the work and allocate resources, and 3) initiate and monitor the task execution. These operations need not be executed in a purely serial fashion but may take a more complex form with executive control operations executed simultaneously or concurrently with task execution.

Examination of the basic operations in further detail (Figure 8) reveals some of the variations possible in the handling of work requests. The following two steps exist in the information gathering phase: 1) collecting information about resource requirements for the work request and 2) identifying the resources available for satisfying those requirements. Information gathering is followed by the selection of a plan for distributing the work and the actual allocation of the resources. If this operation is not successful, three alternatives are available. First, more information on resource availability can be gathered in an attempt to formulate a new work distribution. Further information may be available because a change may have occurred in the status of some resources since the original request for availability information or complete resource information may not have been requested on the initial inquiry. Second, more information can be gathered as above, but the requester now indicates a willingness to "pay more" for the resources. This is referred to as bidding to a higher level. Finally, it may

be necessary to inform the user that it is impossible to satisfy the work request at this time.

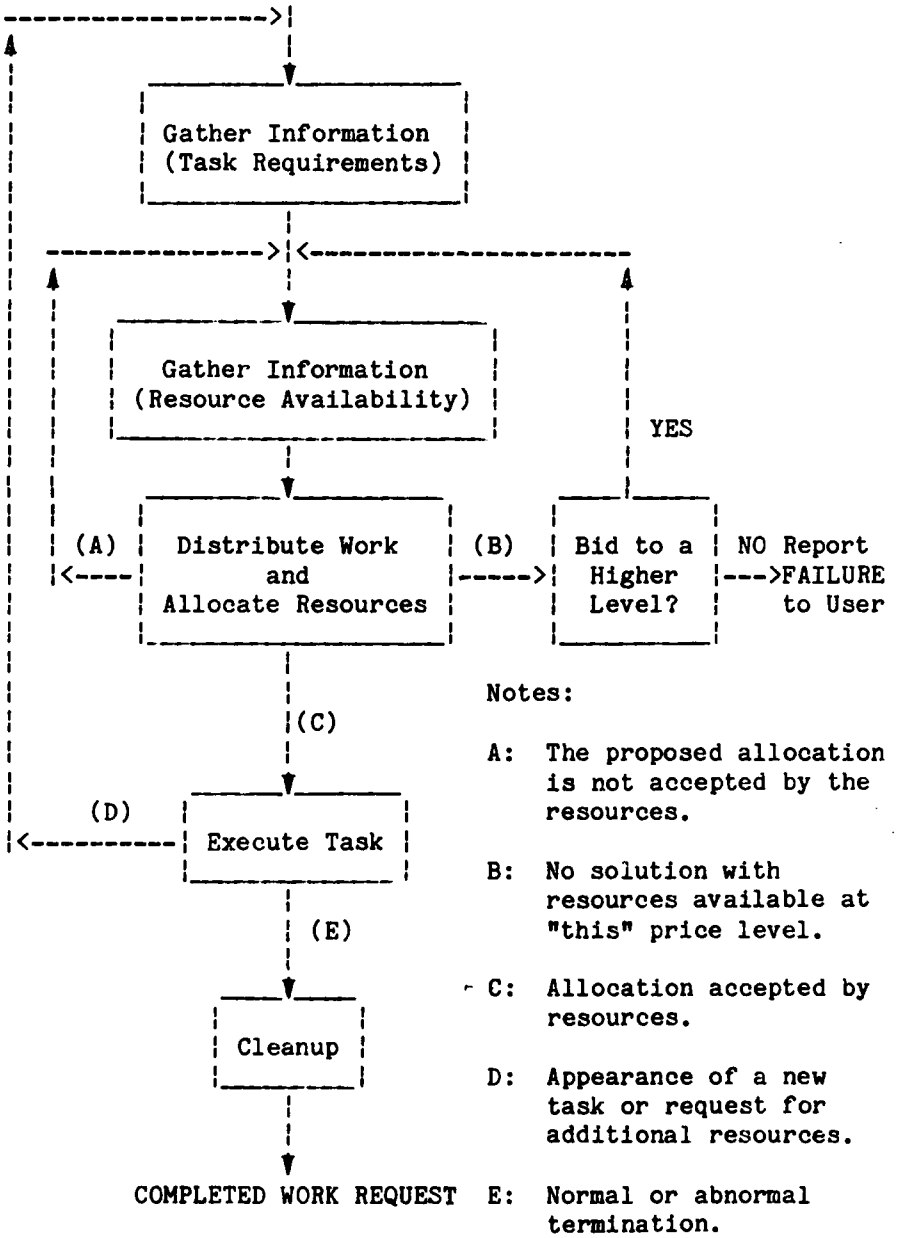


Figure 8. Work Request Processing (Detailed Steps)

3.5.1 Information Gathering

Upon receiving a work request, the first task of the control is to discover what resources are needed to satisfy the work request (Figure 9) and which resources are available to fill these needs (Figure 10). Each work request includes a description of a series of tasks and the connectivity of those tasks. Associated with each task is a series of files. One is distinguished as the execution file and the remainder are input/output files. The executive control must first determine which files are needed. It then must examine each of the execution files to determine the nature of its contents (executable code or commands). Each task will need a processor resource, and those tasks containing command files will also require a command interpreter.

An FDPS executive control must also determine which of the system resources are available. For nonpreemptable resources, the status of a resource can be either "available," "reserved," or "assigned." A reservation indicates that a resource has been promised for possible use by another task sometime in the future and that it should not be given to another user. Typically there is a time-out associated with a reservation that results in the automatic release of the reservation if an actual assignment is not made within a specified time interval, thus freeing resources which otherwise would have been left unavailable by a lost process. The process may be lost because it failed, its processor failed, or the communication link to the node housing the particular resource failed. An assignment, on the other hand, indicates that a resource is dedicated to a user until the user explicitly releases that assignment or termination procedures are executed. Preemptable resources may be accessed by more than one concurrent user and, thus, can be treated in a different manner. For these resources, the status may be indicated by continuous values (e.g., values representing the level of resource utilization) rather than the discrete values described above.

3.5.2 Work Distribution and Resource Allocation

The FDPS executive control must determine the work distribution and the allocation of system resources (Figure 11 & 12). This process involves choosing from the available resources those that are to be utilized. This decision is designed to achieve several goals such as load balancing, maximum throughput, or minimum response time. A general discussion of this problem can be

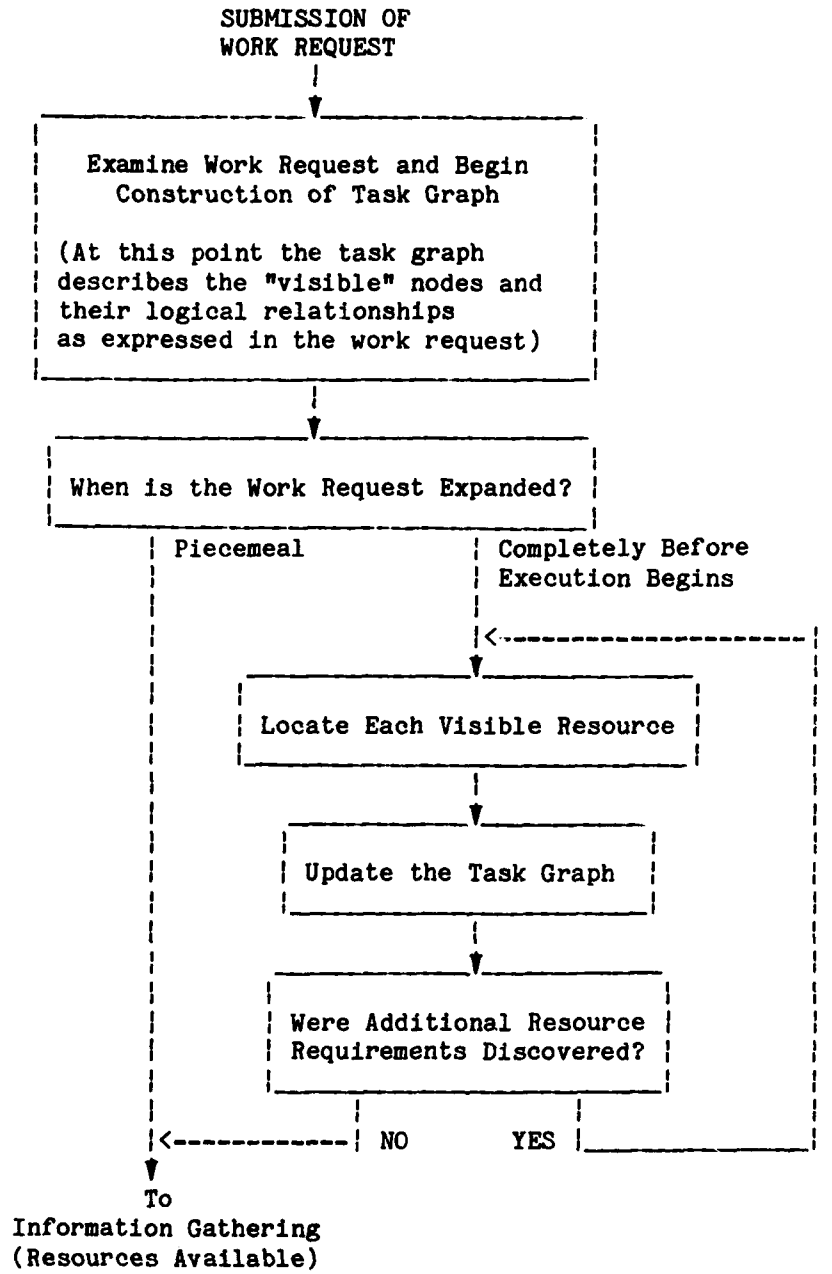
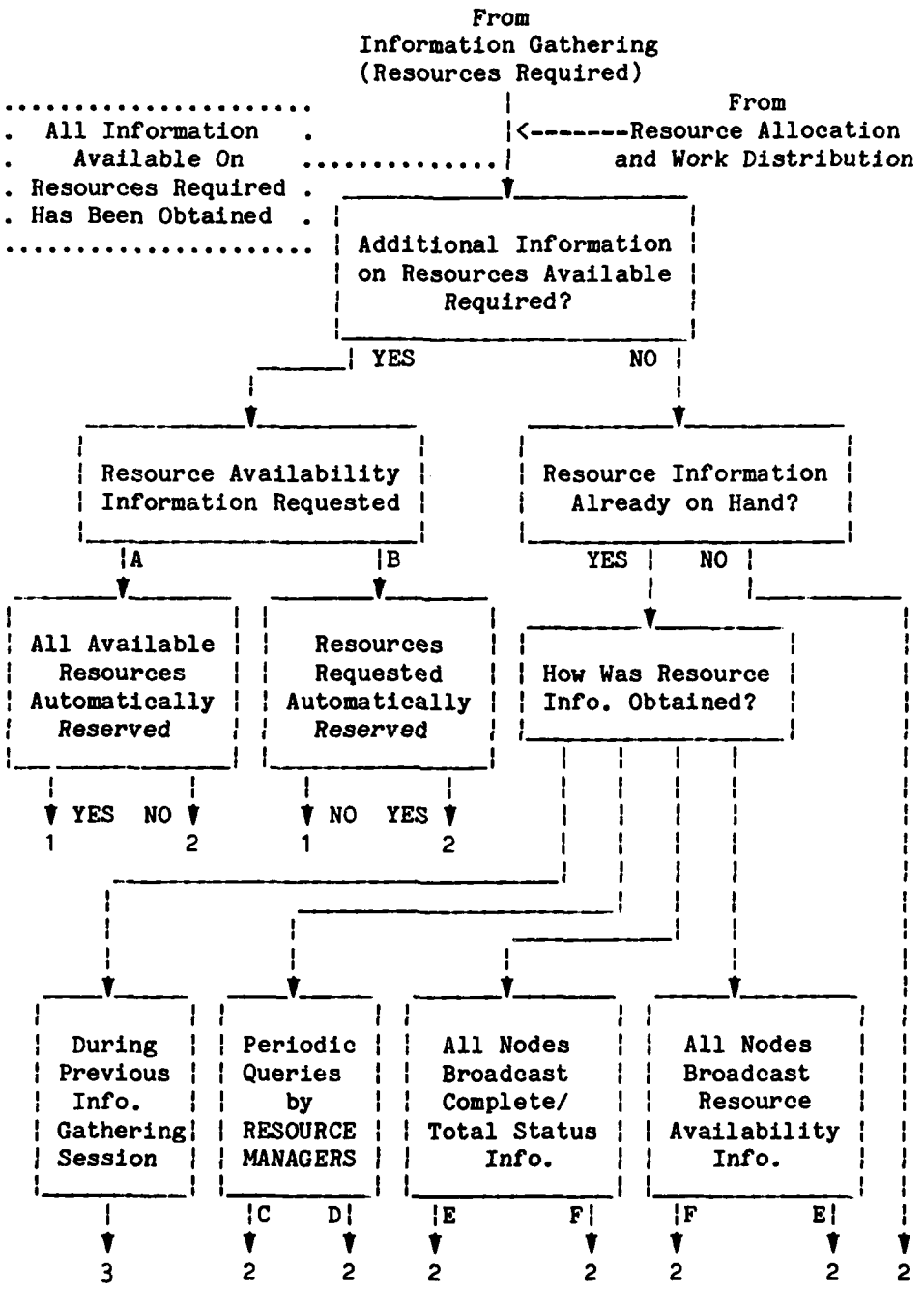


Figure 9. Information Gathering (Resources Required)



(continued on next page)

Figure 10. Information Gathering (Resources Available)

LEGEND AND NOTES

1: Resources Reserved During Information Gathering
2: No Resources Reserved
3: Some Resources May Be Reserved
A: General, for all resources
B: To meet specific task/job requirements
C: Replies cover information on resources available only
D: Replies cover information on the total status
E: Broadcast only significant changes
F: Periodic broadcasts at regular intervals

Figure 10. Information Gathering (Resources Available) (continued)

found in [Chu80], which describes a number of approaches to the problem including graph theoretic, integer programming, and heuristic. A presentation of a graph theoretic approach can be found in [Ston78]. Sharp [Shar81] describes three heuristic algorithms which were developed specifically for Fully Distributed Processing Systems. The first algorithm attempts to minimize the network communication required to satisfy a user work request. Processor load balancing is attempted with the second algorithm. The third algorithm represents a combination of the first two algorithms. This algorithm attempts to minimize communication while also attempting to evenly distribute work across all nodes. All of the preceding methods assume that the work distribution and resource allocation decision is made prior to the start of execution of the processes being scheduled. Bryant [Brya81] proposes that load balancing be accomplished by moving tasks which are already executing. This is accomplished by forming processor pairs via a pairing algorithm and moving tasks from the busier processor to its partner in the processor pair.

Once an allocation has been determined, the chosen resources are allocated and the processes comprising the task set are scheduled and initiated. If a process cannot be immediately scheduled, it may be queued and scheduled at a later time. When it is scheduled, a process control block and any other execution-time data structures must be created.

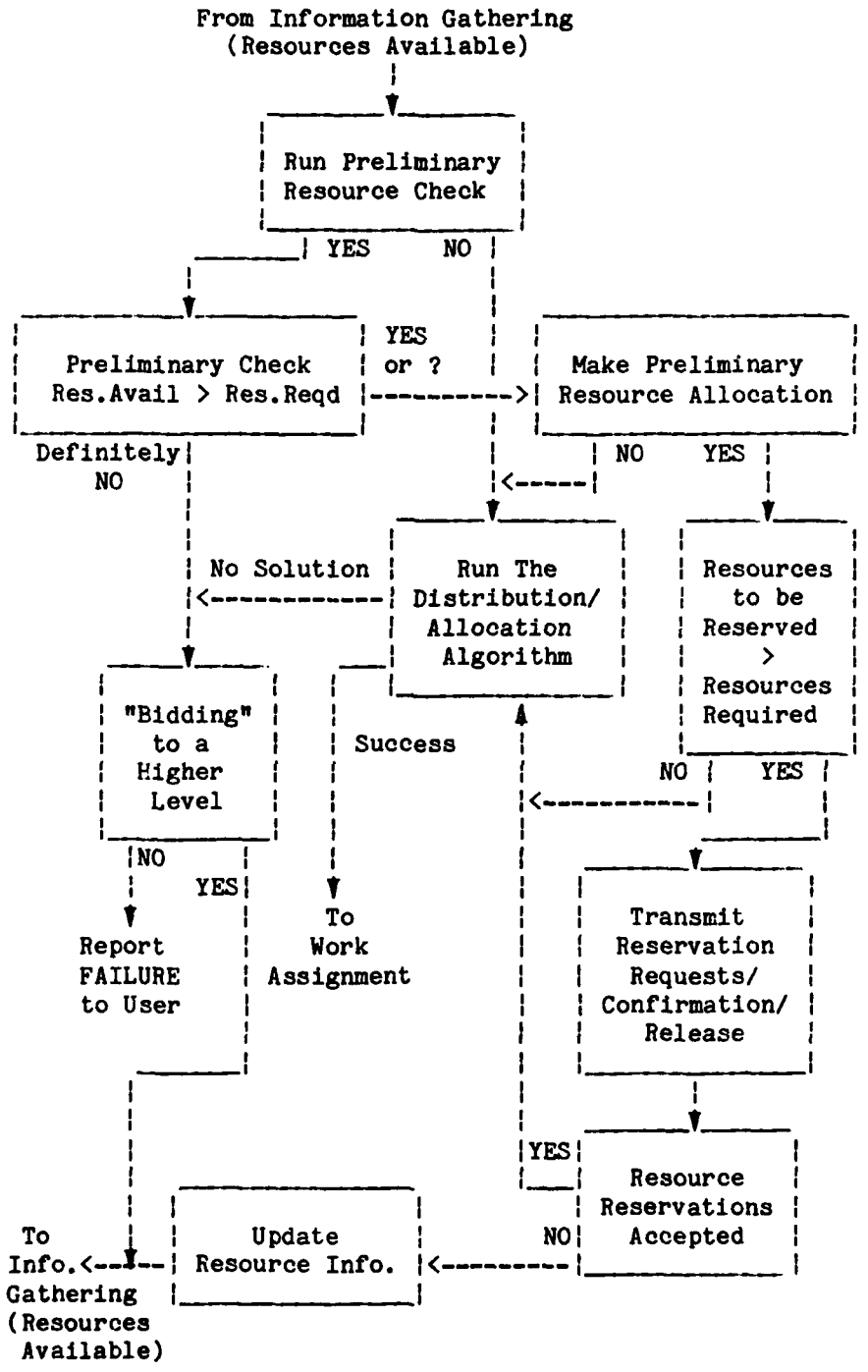


Figure 11. Resource Allocation and Work Distribution

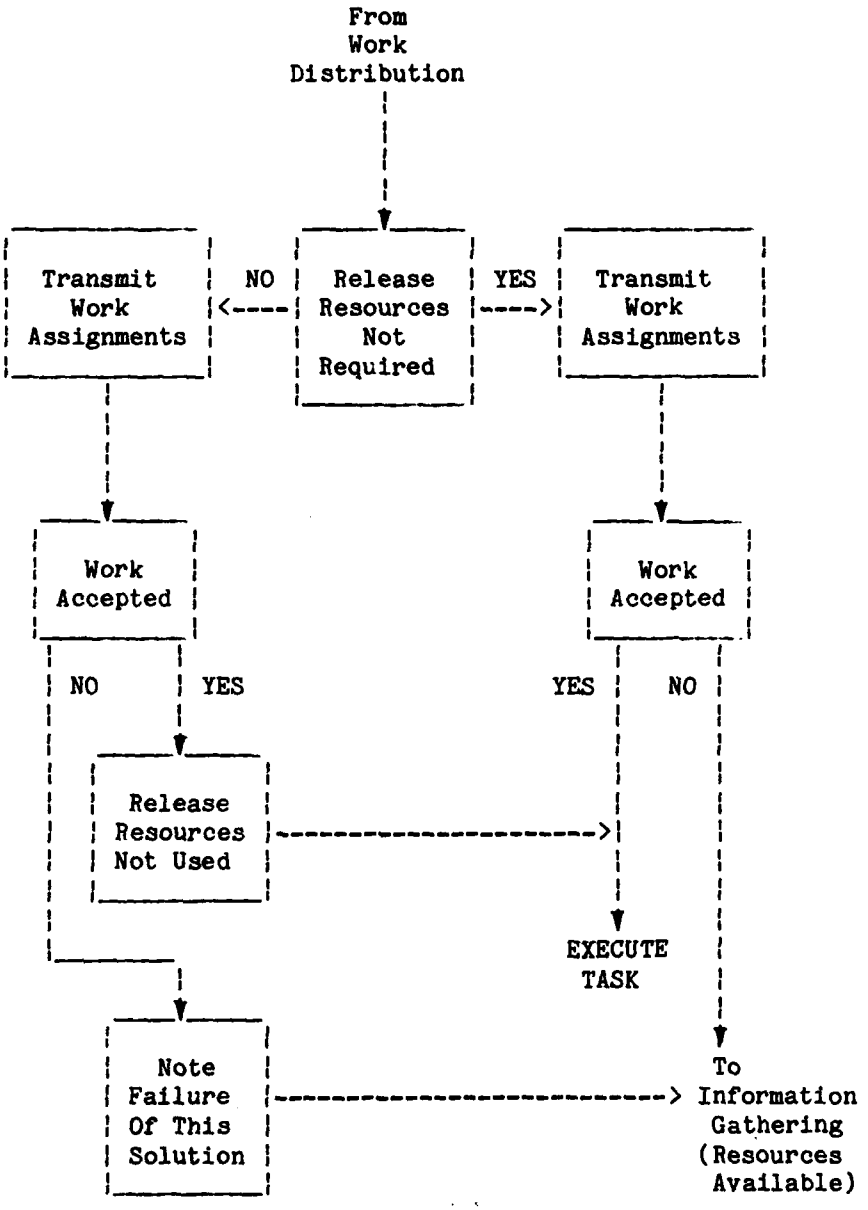


Figure 12. Work Assignment

3.5.3 Information Recording

Information is recorded as a result of management actions and provides a means of maintaining an historical record or audit trail of system activity.

The information recording resulting from management actions records the system state and provides information for decision making. The historical information is useful in monitoring system security as well as its actual performance. It provides a means of examining past activity on a system in order to determine if a breach of security has occurred or how a particular problem or breach of security may have occurred.

Management information is maintained in various structures, including the task graph. The task graph is used to maintain information about the structure of an individual work request, and thus its contents change as processing of the work request proceeds. A task graph is first created when a work request arrives. From that point until the work request is completed, this structure is in a state of dynamic change. It is used to record information about the availability of resources pertinent to this work request and maintains a record of the progress of the various tasks of the work request.

Much of the information contained in the task graph is applicable to historical records. The task graph can be used to house historical information as it is gathered during work request processing. Upon completion of the work request, the historical information is extracted and entered into the permanent historical file. Alternatively, the historical file can be created directly while skipping the intermediate task graph structure.

3.5.4 Task Execution

Finally, an executive control must monitor the execution of active processes. This includes providing interprocess communication, handling requests from active processes, and supervising process termination. The activities associated with interprocess communication include establishing communication paths, buffering messages, and synchronizing communicating processes. The latter activity is necessary to protect the system from processes that flood the system with messages before another process has time to absorb the messages. Active processes may also make requests to the executive control. These may take the form of additional work requests or requests for additional resources. Work requests may originate from either command files or files containing executable code.

The executive control must also detect the termination of processes. This includes both normal and abnormal termination. After detecting process

termination, it must inform processes needing this information that termination has occurred, open files must be closed, and other loose ends must be cleaned up. Finally, when the last process of a work request has terminated, the executive control must inform the originator of the work request of the completion of the processing of his request.

3.6 Variations in FDPS Control Models

There exist an extremely large number of features by which variations in distributed control models can be characterized. Of these, only a few basic attributes seem to deserve attention. These include the nature of how and when a task graph is constructed, the maintenance of resource availability information, the allocation of resources, process initiation, process monitoring, and process termination. In this section, these issues are examined; but since the number of variations possible in each issue are rather large, only those choices considered significant are discussed. Table 1 contains a summary of the problems that have been identified and possible solutions (significant and reasonable solutions) to these problems.

3.6.1 Task Graph Construction

The task graph is a data structure used to maintain information about the applicable task set. The nodes of a task graph represent the tasks of the task set, and the arcs represent the connectivity or flow of information between tasks. There are basically four issues in task graph construction: 1) who builds a task graph, 2) what is the basic structure of a task graph, 3) where are the copies of a task graph stored, and 4) when is a task graph built.

There are three basic alternatives for which component or components will construct the task graph. First, a single "central" node can be responsible for the construction of task graphs for all work requests. Another choice utilizes the control component on the node receiving the work request to construct the task graph. Finally, the job of building the task graph can be distributed among several components. In particular, the nodes involved in executing individual tasks of the work request can be responsible for constructing those parts of the task graph that they are processing.

Table 1. Variations in Control Models

TASK GRAPH CONSTRUCTION:

Who builds the task graph?
1. A central node specializing in task graph building.
2. The node intially receiving and analyzing the work request.
3. All nodes involved in executing the work request.

What is the nature of the task graph?
1. A single complete structure.
2. Multiple structures each consisting of a subgraph.
3. Multiple structures each consisting of a subgraph with one copy of the complete task graph.

Where is the task graph stored?
1. A central node.
2. The node intially receiving and analyzing the work request.
3. A node determined to be in an optimal location.
4. All nodes involved in executing the work request.

When is the task graph built?
1. Completely prior to execution.
2. Piecemeal during execution.

RESOURCE AVAILABILITY INFORMATION:

Who maintains this information?
1. A single central node.
2. All nodes maintain common information.
3. Resources are partitioned with a separate manager for each partition.

Where is the information maintained?
1. At a central node.
2. Separate pieces of information concerning a particular resource type may be kept on different nodes.
3. In multiple redundant copies.
4. Information concerning a particular resource type is kept on a specially designated node.

ALLOCATION OF RESOURCES:

How is concurrency control provided?
1. None is provided.
2. Reservations are used prior to a work distribution decision and then allocated by a lock.
3. Allocated by a lock after the work distribution decision.
4. Resources are locked before the work distribution decision is made.

(continued on next page)

Table 1. Variations in Control Models
(continued)

PROCESS INITIATION:

How is responsibility distributed?
1. Single manager.
   a. Central component for all processes.
   b. Individual components for each work request.
2. Hierarchy of managers.
   a. Two-level hierarchy.
   b. N-level hierarchy.
3. Autonomous managers.

How is refusal of a request to execute a process by a node handled?
1. After repeated attempts, the request is abandoned.
2. After repeated attempts, a new work distribution is obtained.

PROCESS MONITORING:

What type of interprocess communication is provided?
1. Synchronized communication.
2. Unsynchronized communication.

How are task graphs resulting from additional work requests handled?
1. The new task graph is made part of the old one.
2. The new task graph is kept separate.

PROCESS TERMINATION:

Options selected here are determined by those selected for PROCESS INITIATION.

The general nature of the task graph itself provides two alternatives for the design of an executive control. What is of concern is not the content of a task graph but rather its basic structure. One alternative is to maintain a task graph in a single structure regardless of how execution is distributed. The other choice involves maintaining the task graph as a collection of subgraphs with each subgraph representing a part of the work request. For example, a subgraph can represent that portion of the work request that is to be executed on the particular node at which that subgraph is stored.

Another issue of task graph construction concerns where the various copies of the task graph are stored. If the control maintains a task graph as a unified structure representing the complete set of tasks for a work request, this structure may be stored on either a single node or redundant copies may be stored on multiple nodes. The single node can be either a "central" node that is used to store all task graphs, the node at which the original work request arrived (the source node), or a node chosen for its ability to provide this work request with optimal service. If the task graph is divided into several subgraphs, these can be maintained on multiple nodes.

Finally, there is the issue concerning the timing of task graph construction within the sequence of steps that define work request processing. Two choices are available: 1) the task graph can be constructed completely, or at least to the maximum extent possible, before execution is begun, or 2) the task graph can be constructed incrementally as execution progresses.

3.6.2 Resource Availability Information

Another characteristic that distinguishes various control models is the maintenance of resource *availability information.* Of importance is "who maintains this information" and "where is this information maintained." A particular model need not uniformly apply the same technique for maintaining resource availability information to all resources. Rather, the technique best suited to a particular resource class may be utilized.

The responsibility for maintaining resource availability information can be delegated in a variety of ways. The centralized approach involves assigning a single component this responsibility. Requests and releases for resources flow through this specialized component which maintains the complete resource availability information in one location.

A variation of this technique maintains complete copies of the resource availability information at several locations. This technique is similar to that used in the ARAMIS Distributed Computer System [Caba79a,b]. Components at each of these locations are responsible for updating their copy of the resource availability information in order to keep it consistent with the other copies. This requires a protocol to insure that consistency is maintained. For example, two components should not allocate a file for writing to different users at the same time. The ARAMIS Distributed Computer

System provides such a protocol. The nodes of the network are organized in a logical loop. A message called the control vector (CV) circulates about the loop. The holder of the CV may allocate or deallocate resources. The updates to the resource data base are packaged in a message called the update vector (UPV). The UPV is passed around the loop allowing each node the opportunity to bring its resource data base into a state consistent with the other nodes. When the holder of the CV receives the UPV it sent, the CV is sent on to the next node.

Another approach exhibiting more decentralization requires dividing the collection of resources into subsets or classes and assigning separate components to each subset. Each component is responsible for maintaining resource availability information on a particular subset. In this case, requests for resources can be serviced only by the control component responsible for that particular resource. Resources may be named in a manner such that the desired manager is readily identifiable. Alternatively, a search may be required in order to locate the appropriate manager. This search may involve passing the request from component to component until one is found that is capable of performing the desired operation.

Preemptable resources, which can be shared by multiple concurrent users (e.g., processors and communication lines), do not necessarily require the maintenance of precise availability information. For these resources, it is reasonable to maintain only approximate availability information because such resources are rarely exhausted. The primary concern in this instance is degraded performance. Therefore, a good estimate of resource utilization is needed.

3.6.3 Allocating Resources

One of the major problems experienced in the allocation of resources is concurrency control. In a hospitable environment, it is possible to ignore concurrency control. The users are given the responsibility of insuring that access to a shared resource such as a file is handled in a consistent manner. In other environments, such as that presented by an FDPS, concurrency control is an important issue. In an FDPS, the problem is even more difficult than in a centralized system due to the loose coupling inherent in the system.

There are basically three approaches to solving the problem of concurrent requests for shared resources. In the first approach resources are reserved at the time of information gathering. The reservation prevents other users from acquiring the resource and is effective for only a limited period, a period long enough to make a work distribution decision and allocate the resources determined by the decision. The other two solutions to this problem do not use reservations. In one case a lock instead of a reservation is applied prior to the formulation of the work distribution decision. This requires the explicit release of all resources not needed. The reservation provides the control with further information as to the status of the resource. A reservation means that the resource may be used in the near future by a process. Therefore, reserved resources can be distinguished from locked resources. The last technique attempts the formulation of a work distribution decision without reserving or locking resources. If resources cannot be allocated, the executive control must either wait until they can be allocated or attempt a new work distribution.

3.6.4 Process Initiation

Several issues arise concerning process initiation. Of primary interest is the distribution of responsibility. Responsibility can be organized in numerous ways but the following three organizations appear to be the most popular and the most promising: a single manager, a hierarchy of managers, or a collection of autonomous managers. Two approaches result from the single manager concept. In the first organization, a central component is in charge of servicing all work requests and controlling the processes resulting from these work requests. All decisions concerning the fate of processes and work requests are made by this component. A variation of this organization assigns responsibility at the level of work requests. Each work request has its own separate manager making all decisions concerning the fate of the work request and its processes.

Management can also be organized in a hierarchical manner. There are a variety of ways hierarchical management can be realized, but in this dissertation, only two, the two-level hierarchy and the n-level hierarchy, are discussed. The two-level hierarchy has at the top level a component that is responsible for an entire work request. At the lower level are a series of components each responsible for an individual task of the work request. The

lower level components take direction from the high level component and provide results to the higher level. The n-level hierarchy utilizes in its top and bottom levels the components described for the two-level hierarchy. The middle levels are occupied by components that are each responsible for a subgraph of the entire task graph. Therefore, a middle component takes direction from and reports to a higher level component which is in charge of the part of the task graph that includes the subgraph for which the middle component is responsible. The middle component also directs lower level components, each of which are responsible for a single task.

Another organizational approach utilizes a series of autonomous management components. Each component is in charge of a subset of the tasks of a work request. Cooperation between the components is required in order to realize the orderly completion of a work request.

Regardless of the organization, at some point a request for the assumption of responsibility by a component will be made. Such a request may be reasonably denied for two reasons: 1) the component does not possess enough resources to satisfy the request (e.g., there may not be enough space to place a new process on an input queue), or 2) the component may not be functioning. The question that arises concerns how this denial is handled. One solution is to keep submitting the request either until it is accepted or until a certain number of attempts have failed. If the request is never accepted, the work request is abandoned, and the user is notified of the failure. Instead of abandoning the work request, it is possible that a new work distribution decision can be formulated utilizing the additional knowledge concerning the failure of a certain component to accept a previous request.

3.6.5 Process Monitoring

The task of monitoring process execution presents the FDPS executive control with two major problems, providing interprocess communication and responding to additional work requests and requests for additional resources. Interprocess communication is required in order to support the type of work requests envisioned for an FDPS. Recall that these work requests involved the specification of multiple communicating tasks. The question that must be addressed concerns the nature of the communication primitives provided by the FDPS executive control. This question arises due to the variety of communica-

tion techniques being offered by current languages. There are two basic approaches found in current languages, synchronized communication and unsynchronized communication (buffered messages). Synchronized communication requires that the execution of both the sender and the receiver be interrupted until a message has been successfully transferred. Examples of languages utilizing this form of communication are Hoare's Communicating Sequential Processes [Hoar78] and Brinch Hansen's Distributed Processes [Brin78]. In contrast, buffered messages allow the asynchronous operation of both senders and receivers. Examples of languages using this form of communication are PLITS [Feld79], PRONET [LeBl81], and STARMOD [Cook80].

The executive control is required to provide communication primitives that are suitable to one of the communication techniques discussed above. If the basic communication system utilizes synchronized communication, both techniques can be easily handled. The problem with this approach is that there is extra overhead incurred when providing the message buffering technique. Alternatively, if the basic communication system utilizes unsynchronized communication, there will be great difficulty in realizing a synchronized form of communication.

The task of monitoring processes also involves responding to requests generated by the executing tasks. These may be either requests for additional resources (e.g., an additional file) or new work requests. If the new request is a work request, there is a question as to how the new set of tasks is to be associated with the existing set of tasks. The new set could either be included in the existing task graph or a new task graph could be constructed for these new tasks. The former technique allows the component making the work distribution decision for the new work request to consider the utilization of other resources by the control. The latter technique may not allow such a situation to occur.

3.6.6 Process Termination

When a process terminates some cleanup work must be accomplished (e.g., closing files, returning memory space, and deleting records concerning that process from the executive control's work space). In addition, depending on the reason for termination (normal or abnormal), other control components may need to be informed of the termination. In the case of a failure, the task

graph will contain the information needed to perform cleanup operations (e.g., the identities of the processes needing information concerning the failure). Both the nature of the cleanup and the identity of the control components that must be informed of the termination are determined from the design decisions chosen for monitoring task execution.

3.6.7 Examples

To gain a better appreciation for some of the basic issues of control in an FDPS, it is useful to examine several examples of work request processing on an FDPS. In each example, emphasis is placed on the operations involved in the construction of task graphs. In these examples, the work distribution decision assigns the execution of processes to the same nodes that house the files containing their code. The concern of the first eight examples is the impact of variations in work requests on task graph construction. In these examples the various parts of the overall task graph describing the complete work request are stored on the nodes utilized by each part. The last three examples examine three different techniques for storing the task graphs. In the examples (Figures 13 to 23) the following symbols are utilized:

[ ]	visible external reference(s)
{ }	embedded external reference(s)
(n)A	responsibility for A delegated from node n
A(n)	responsibility for A delegated to node n
a-->b	IPC from process a to process b
A,B,...	uppercase letters indicate command files
a,b,...	lowercase letters indicate executable files
u,v,w,x,y,z	indicate data files

The first example (Figure 13) consists of a simple request in which all external references are visible and all required files are present on the source node, the node where the original request arrived. Because the references are visible, the entire task graph can be completed in one step. The second example (Figure 14) is similar to the first except that there is a chained reference utilizing a command file. Again, because all external references are visible before execution, the entire task graph can be completed in one step. This work request can be processed in an alternate manner as shown by the third example (Figure 15) where references are located and linked in a piecemeal fashion, perhaps as the executable files are invoked by the sequence of commands in the command file. Example 4 (Figure 16) adds a slight variation by introducing an explicit interprocess communication (IPC)

definition. The task graph can still be constructed in one step because all references are visible.

The next series of examples consider the impact of locating resources on nodes other than the source node. In example 5 (Figure 17), all the referenced resources reside on a single node other than the source node with the exception of one resource that has redundant copies on two different nodes. Because the resources are not on the source node, negotiation is required to transfer responsibility for a piece of the task graph. In addition, because there is a resource with two redundant copies, a decision as to which to utilize must be made and a negotiation must occur to transfer responsibility. Example 6 (Figure 18) is similar to example 5 and demonstrates the impact of IPC across nodes.

The effect of embedded references is demonstrated in examples 7 and 8. In example 7 (Figure 19), all resources reside on the source node. Multiple steps are required to construct the task graph because all of the resources are not visible and thus cannot be identified until after execution has progressed to the point where the references are encountered. Example 8 (Figure 20) is slightly more complex with resources spread over multiple nodes. Again multiple steps are required because parts of the task graph cannot be constructed until they are referenced during execution. With resources distributed on different nodes, negotiations to assign and accept responsibility must occur.

The last three examples demonstrate three different techniques for storing task graphs. In each example the same work request is utilized. This request has all visible references to resources distributed over multiple nodes. In the first eight examples and example 9 (Figure 21), the parts of the overall task graph are stored on the nodes executing the "root" or "subroot" process. In addition, each subgraph contains a small portion of information linking it to the rest of the overall task graph. Example 10 (Figure 22) maintains these subgraphs on the processing nodes while maintaining a complete task graph at the source node. Example 11 (Figure 23) maintains complete task graphs at all nodes where any processing of the work request occurs. The motivation for the last two techniques in which a large amount of redundant information is maintained is to enhance the ability to

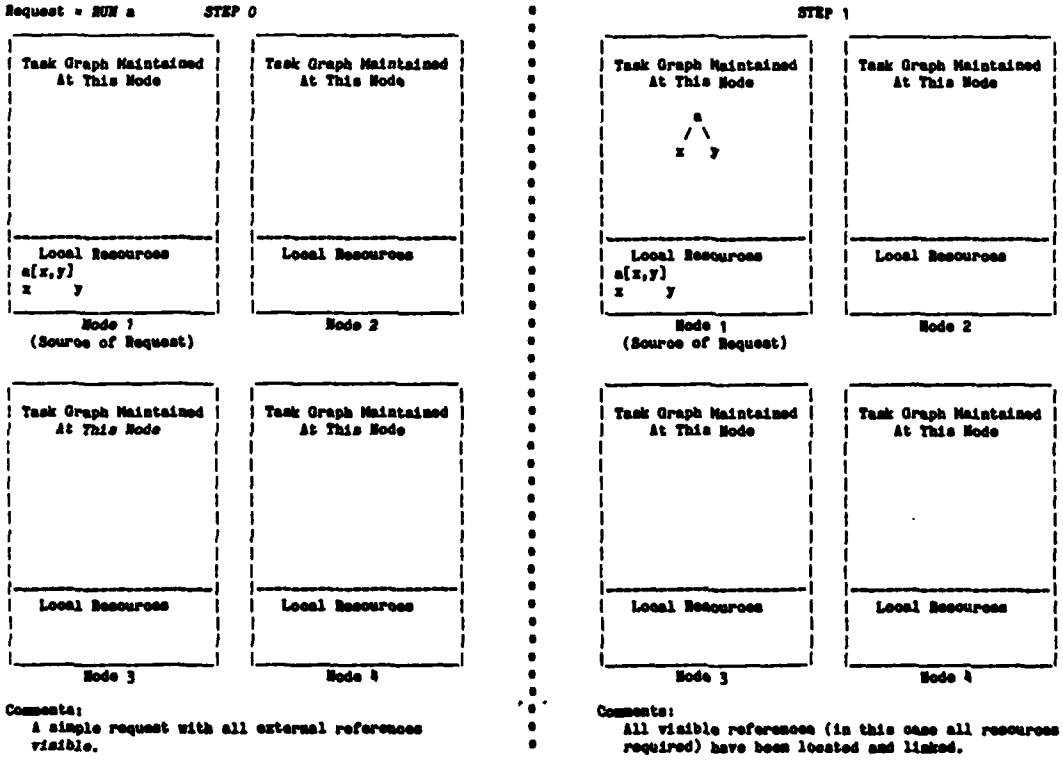


Figure 13. Example 1

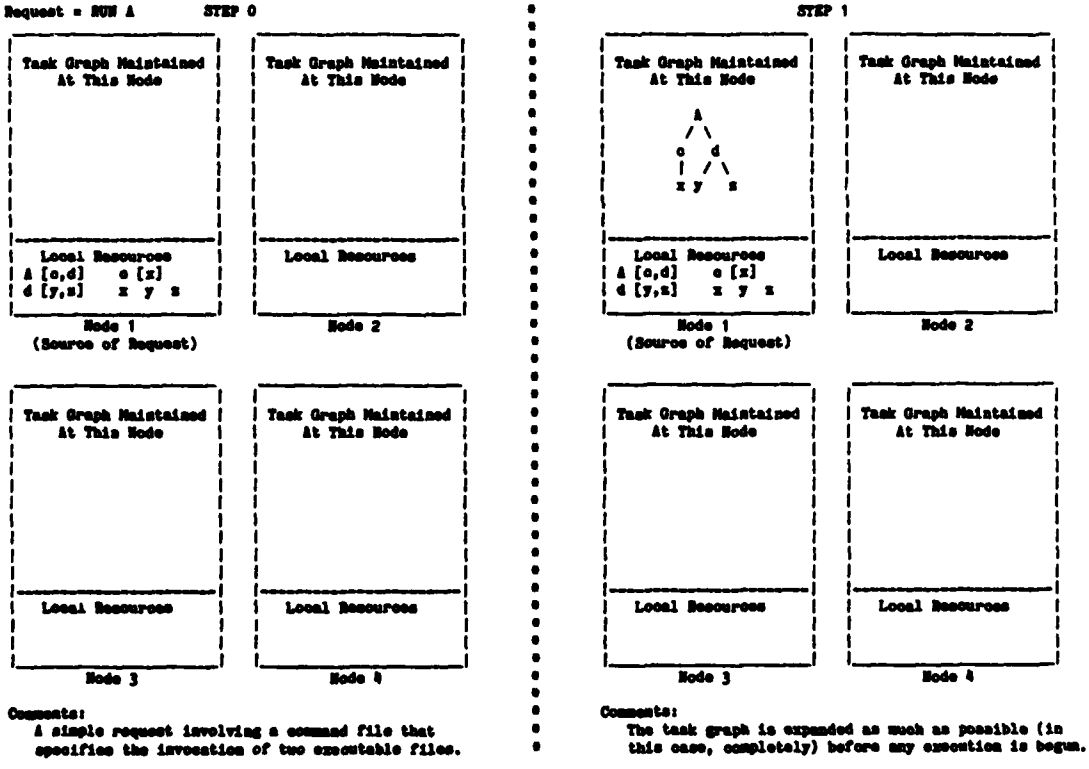


Figure 14. Example 2

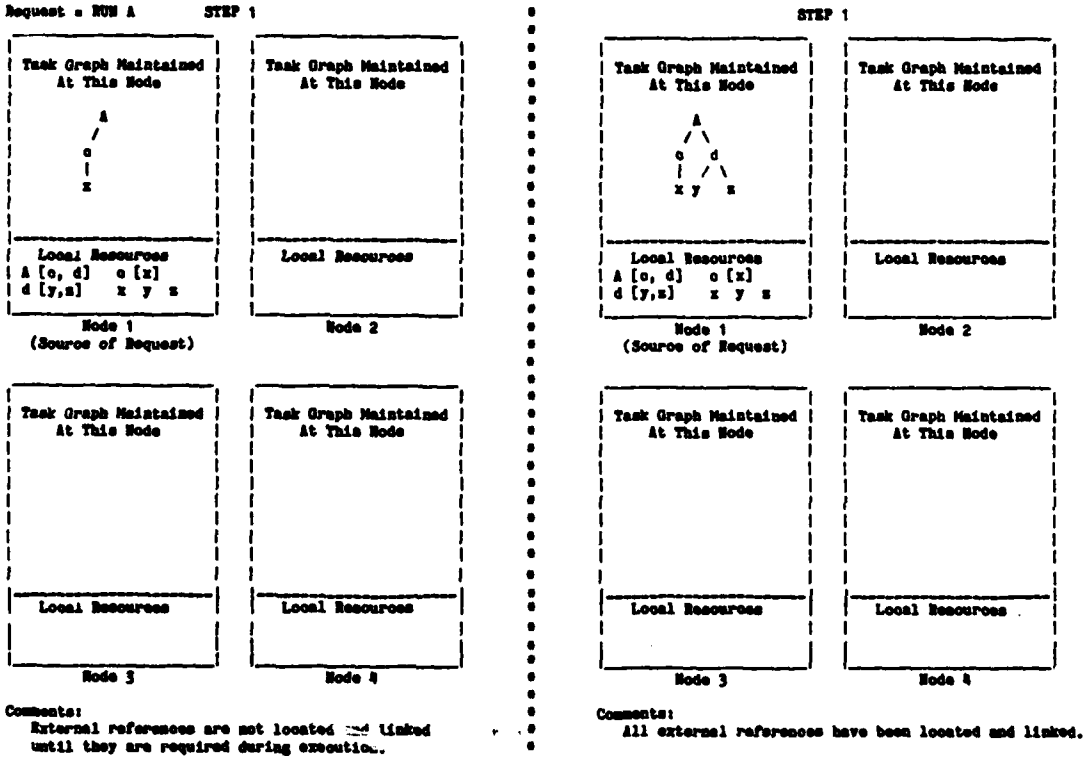


Figure 15. Example 3

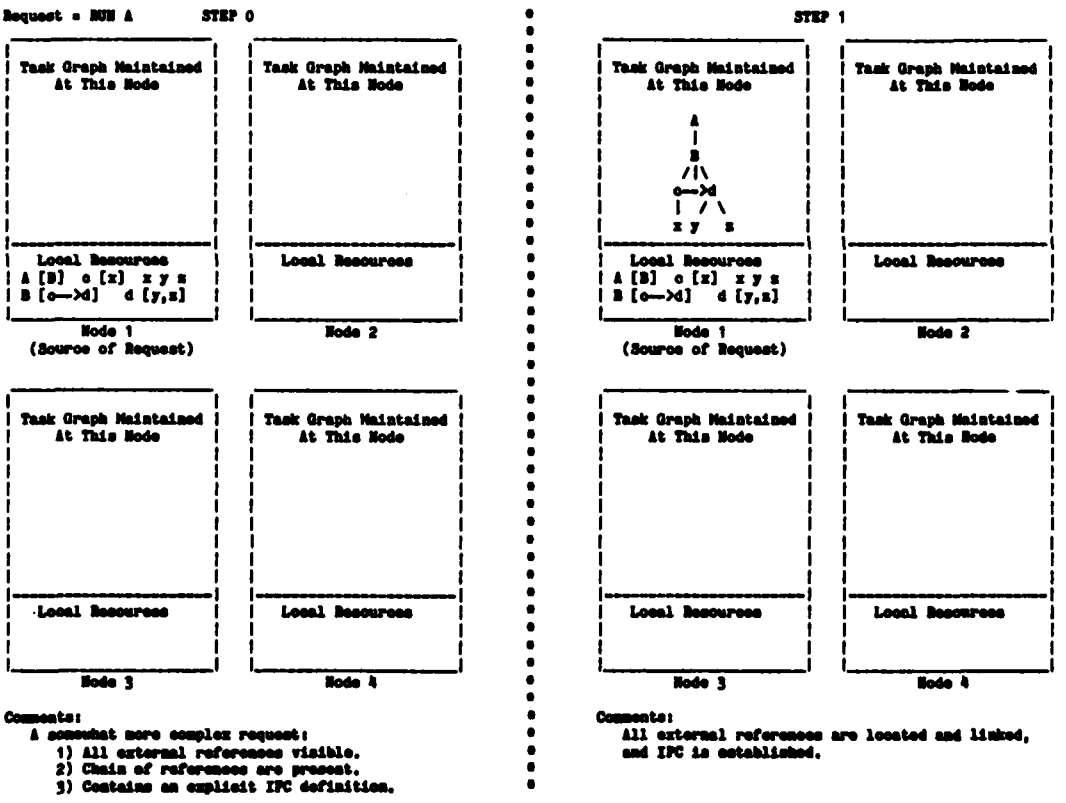


Figure 16. Example 4

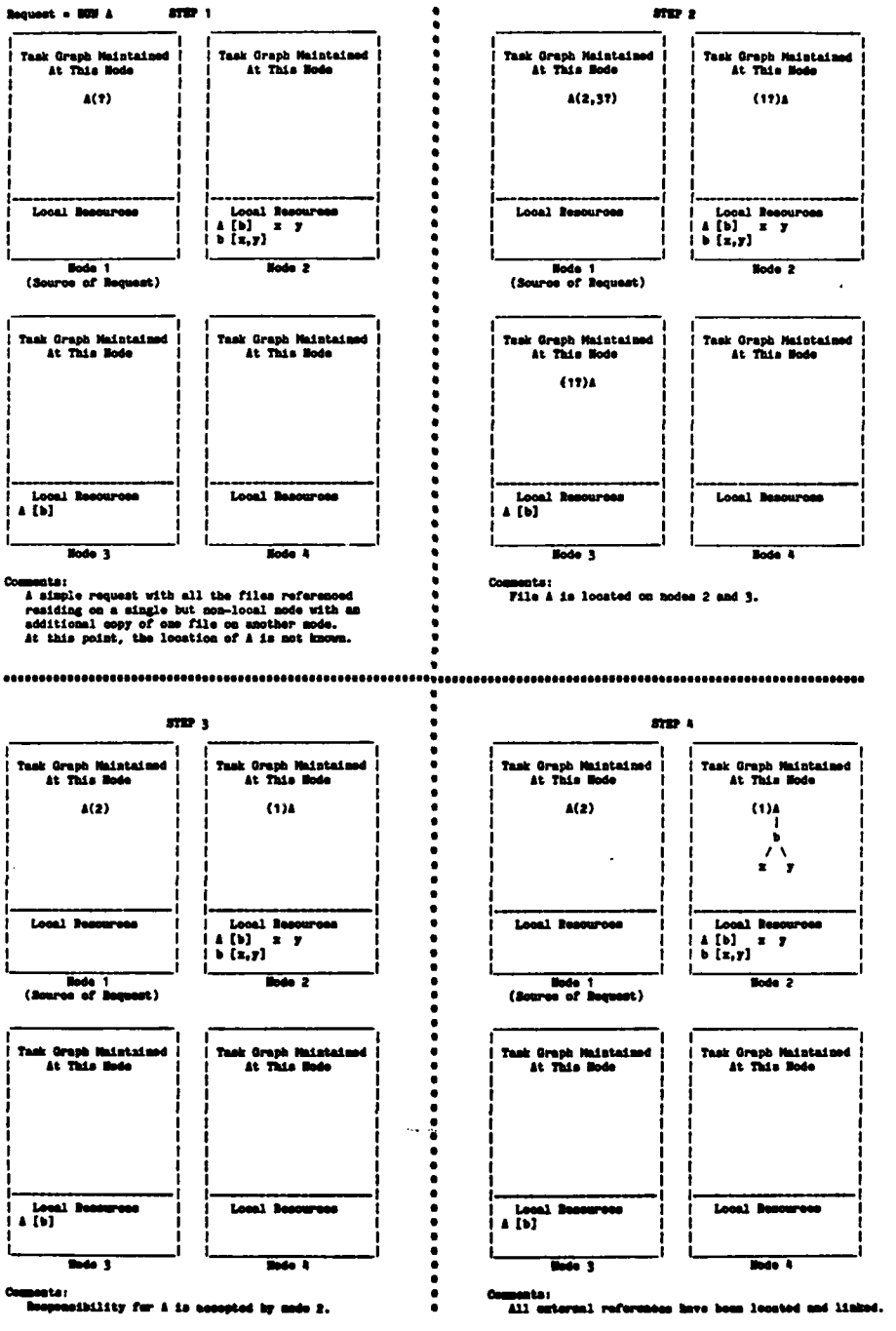


Figure 17. Example 5

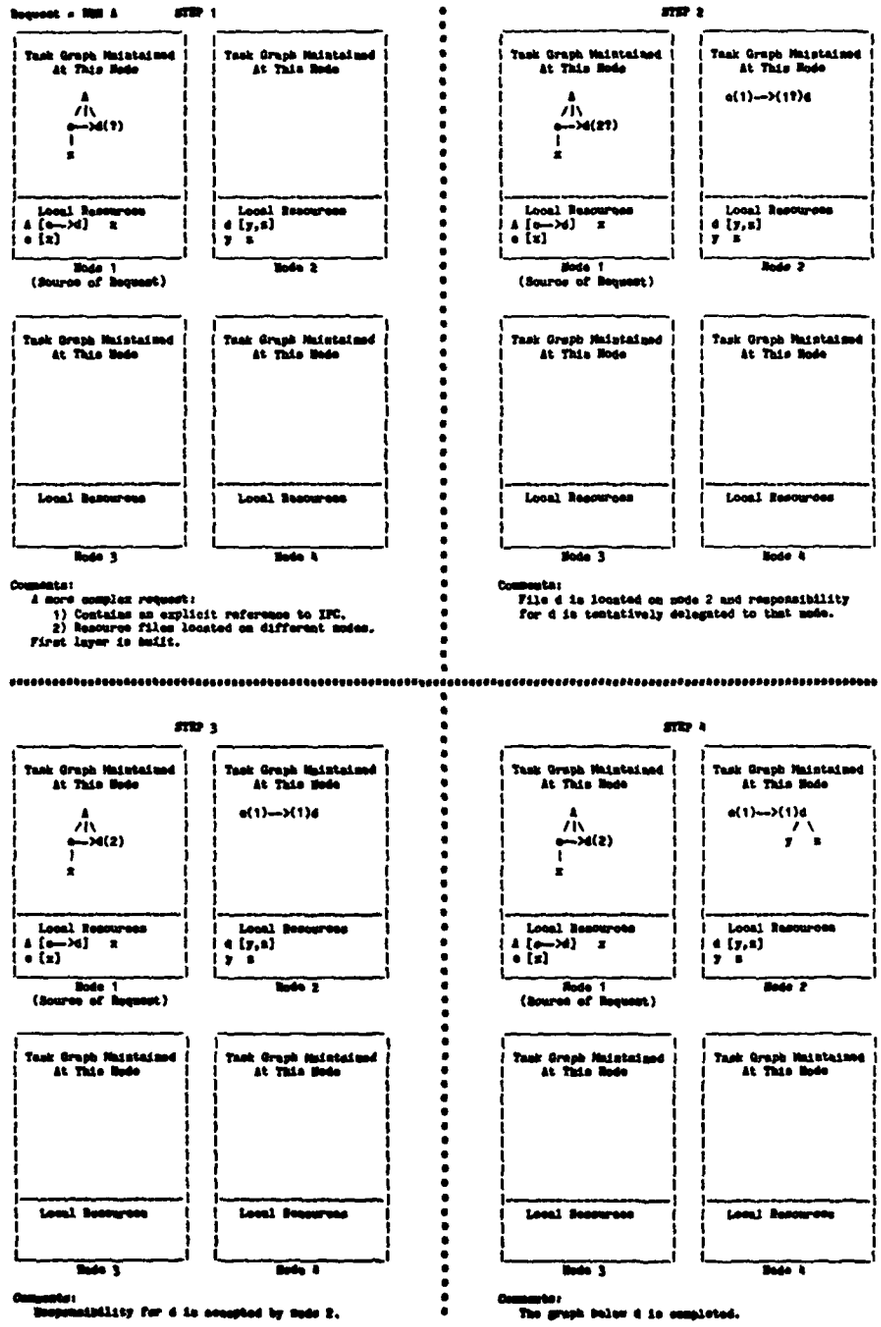


Figure 18. Example 6

recover from failures.

Having studied the construction of task graphs in a broad sense, it is appropriate to examine the details of the task of processing a work request. This is illustrated in two figures. Figure 24 outlines the basic steps involved in work request processing utilizing a particular control strategy. A local search is first made for resources, and a global search is performed only if necessary. An example of the use of this strategy for processing the work request from example 6 (Figure 18) is presented in Figure 25. This example demonstrates how the task graph is progressively constructed as information is obtained.

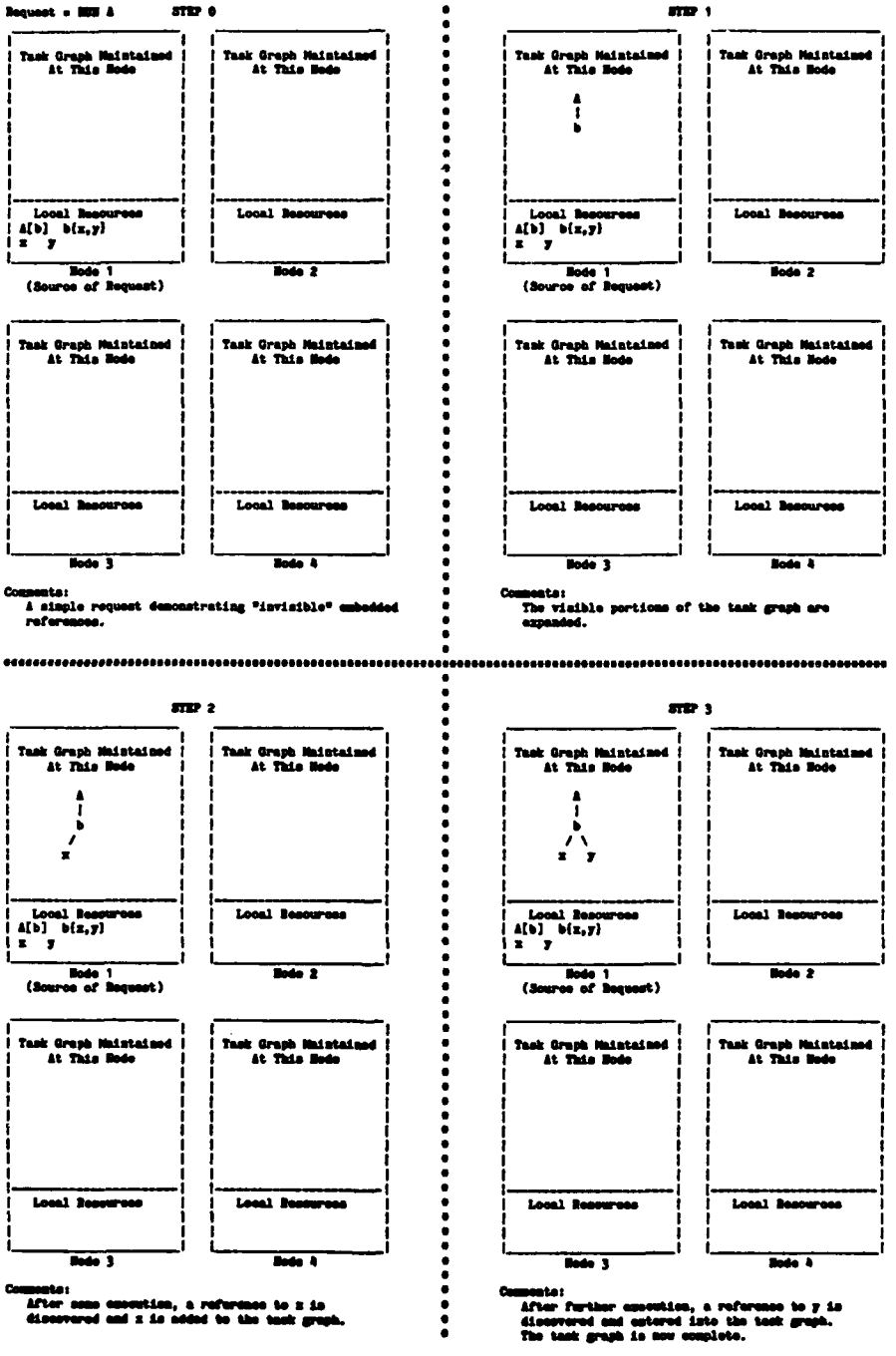


Figure 19. Example 7

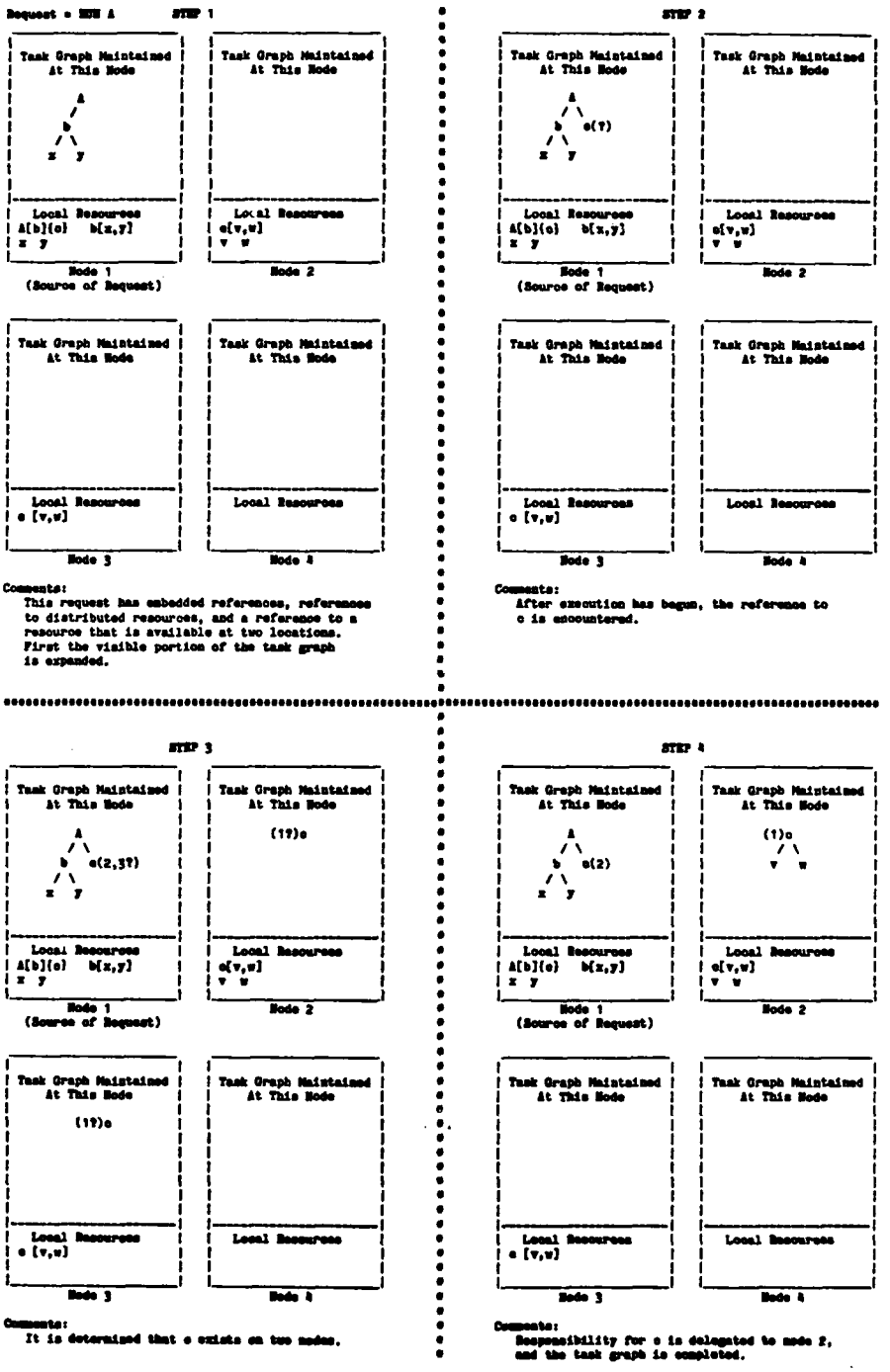


Figure 20. Example 8

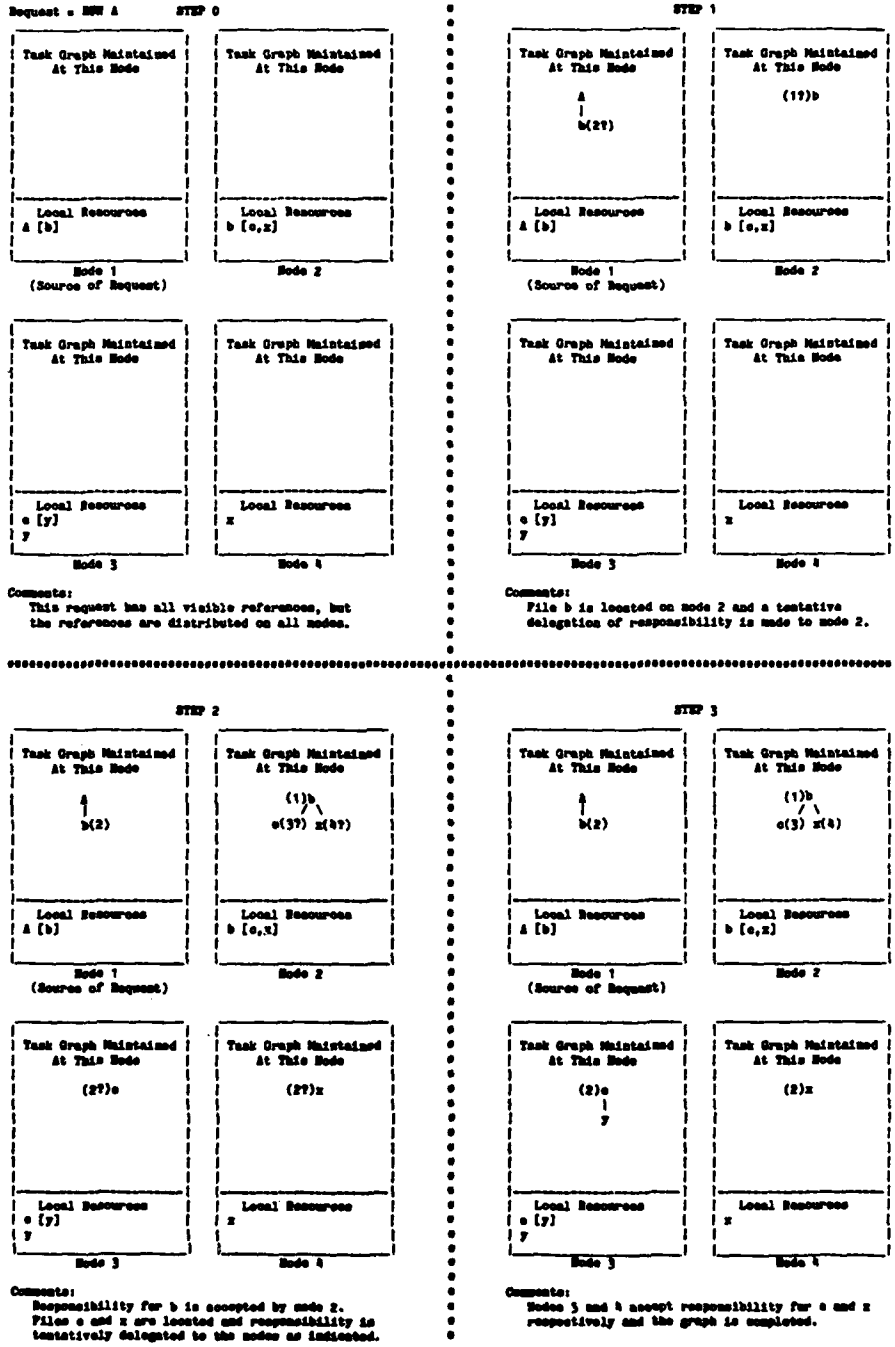


Figure 21. Example 9

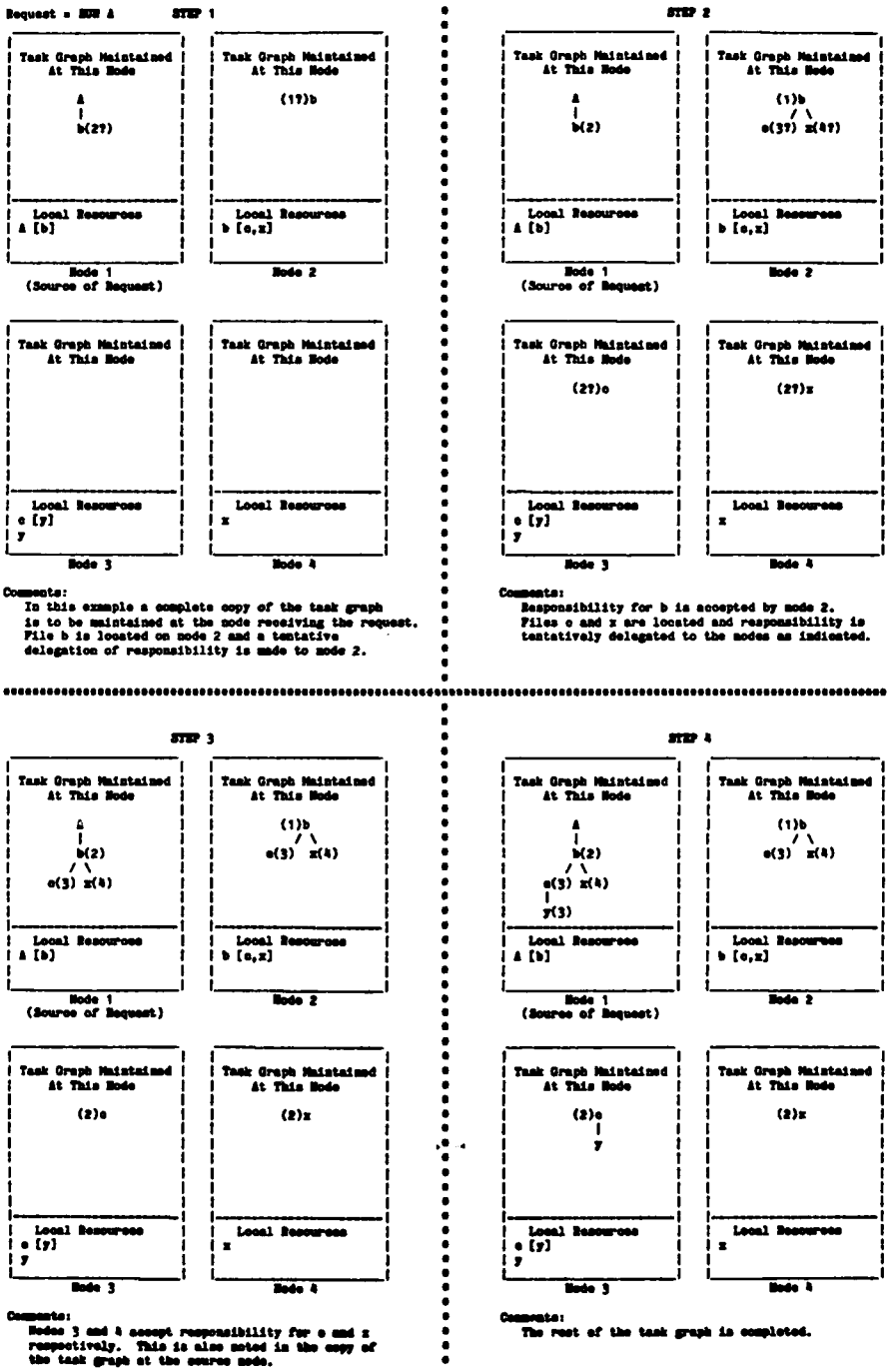


Figure 22. Example 10

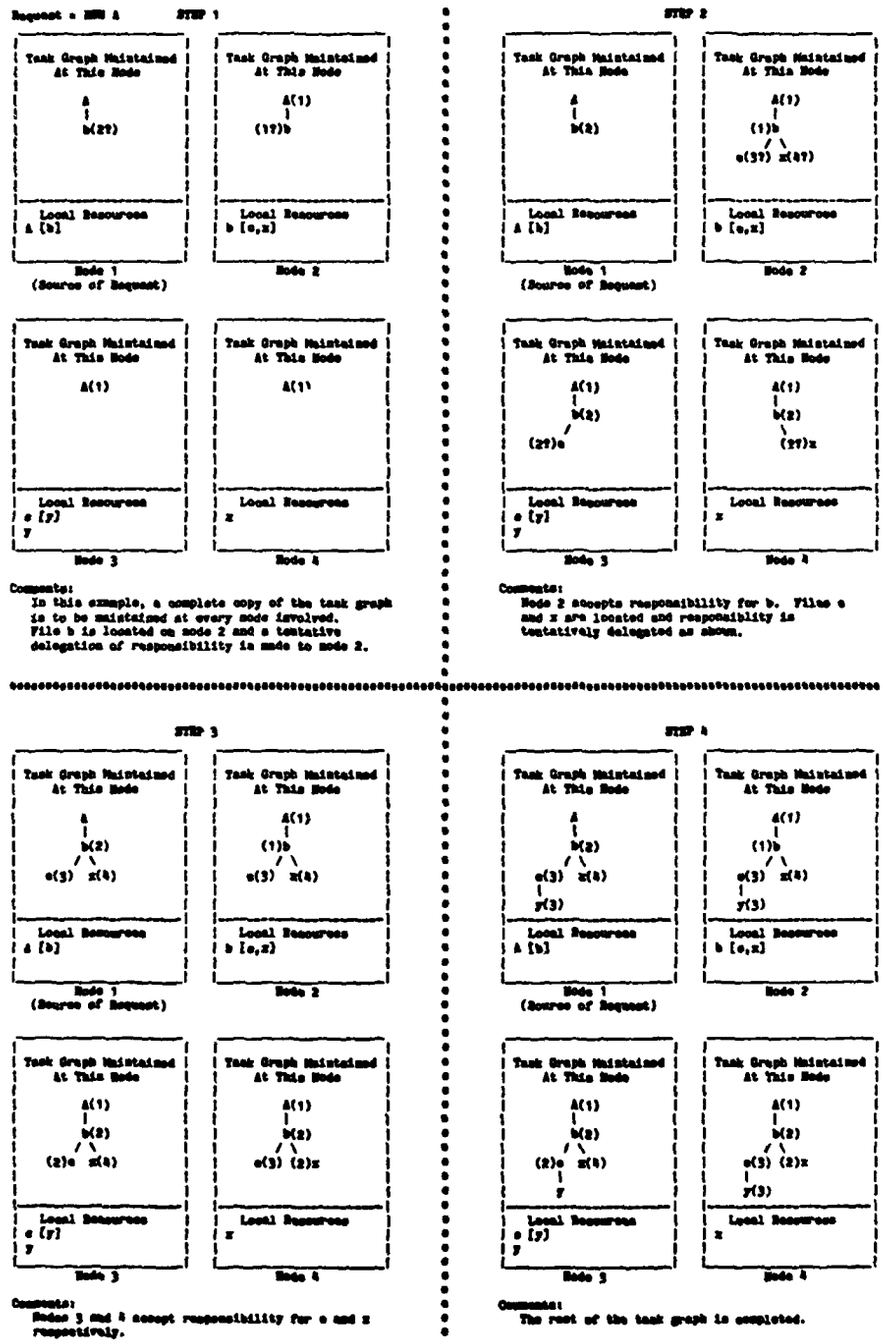


Figure 23. Example 11

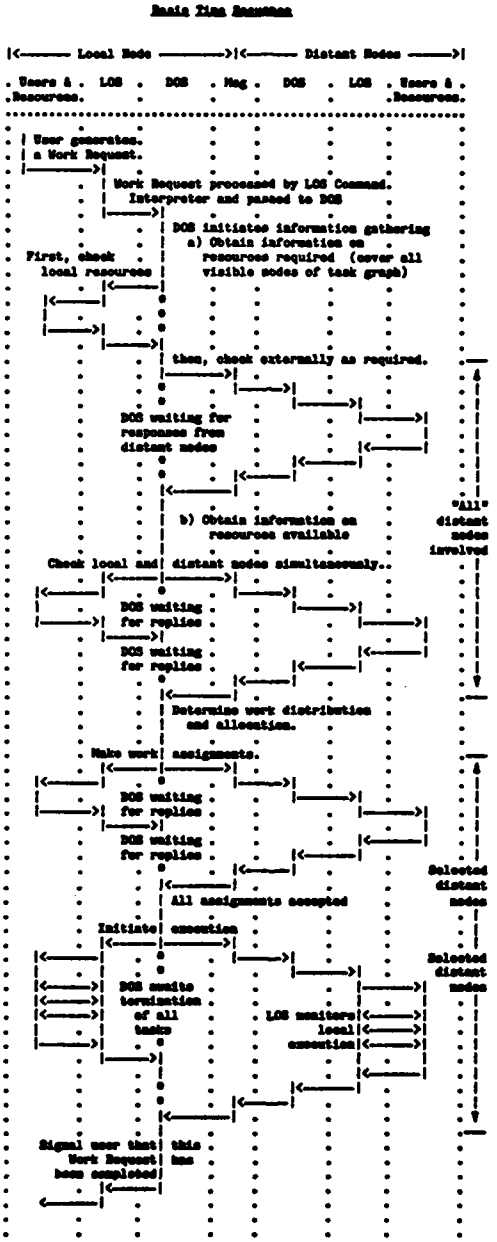


Figure 24. Basic Steps in Work Request Processing

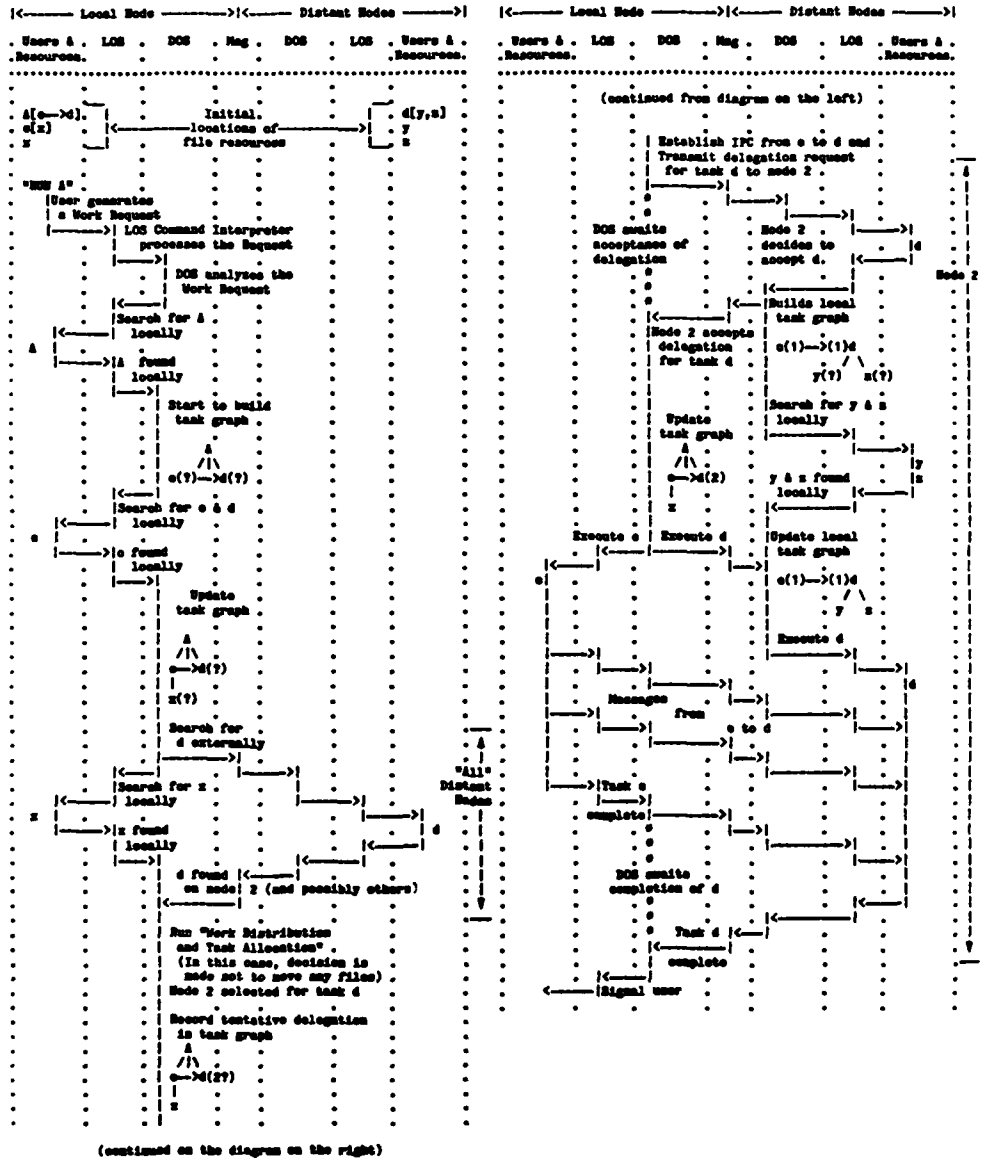


Figure 25. An Example of Work Request Processing

SECTION 4

EXAMPLE CONTROL MODELS

In this chapter six different control models are presented. Pseudo code descriptions of these models appear in the Appendix. The models, named XFDPS.1, XFDPS.2, XFDPS.3, XFDPS.4, XFDPS.5, and XFDPS.6 respectively, demonstrate a wide variety of control strategies.

The first model partitions the system's resources and manages each partition separately. A global search is performed in order to obtain resources for each service request. A centralized directory of all resources is utilized in the second model. All service requests are handled by the control component on the node housing the central directory. The third model is similar to the first differing only in the strategy employed to search for resources. In the third model, a local search is performed first, and a global search is utilized only if the local search fails to provide the necessary resources. Multiple redundant directories are maintained by the fourth model. Control components on each node are activated in a serial fashion in order to control the allocation and deallocation of resources. This strategy is similar to that employed in the ARAMIS Distributed Computer System. Model five is similar to the first model but utilizes a different scheme for reserving resources. The reservation is made following the work distribution and resource allocation decision. Model six investigates the effects of maintaining the task graph as a partitioned unit residing on multiple nodes rather than as a single monolithic data structure.

All of the models presented in this chapter are basically a variation of the first model. Therefore, a detailed description of model XFDPS.1 is presented while the presentation of the remaining models explains only how they differ from the first model. There is a complete pseudo code description of model XFDPS.1 in the Appendix. The remaining models are presented by showing that portion of the code for the model that differs from that for model XFDPS.1.

4.1 The XFDPS.1 Control Model

The XFDPS.1 control model was first defined in [Sapo80] and further refined in [Ensl81a] and [Ensl81b]. With the aid of a simulation environment,

a more thorough definition of this model has been realized. The XFDPS.1 model is composed of six types of components: TASK SET MANAGERs, FILE SYSTEM MANAGERs, FILE SET MANAGERs, PROCESSOR UTILIZATION MANAGERs, PROCESSOR UTILIZATION MONITORs, and PROCESS MANAGERs. (See Figure 26.) The basic strategy of this model of control is to partition the system's resources and assign separate components to manage each partition.

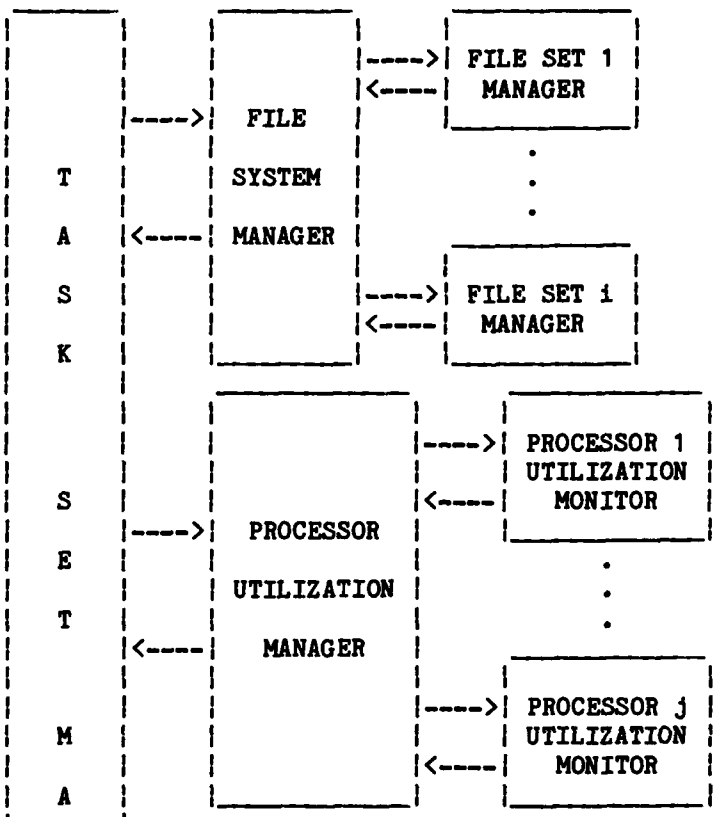
4.1.1 Task Set Manager

A TASK SET MANAGER is assigned to each user terminal and to each executing command file. The name TASK SET MANAGER results from the nature of user work requests, which originate from user terminals and command files. The work requests specify task sets which contain one or more executable files called tasks (these contain either object code or commands) and any input or output files used by the tasks. It is possible for the tasks of a work request to communicate, and this communication (task connectivity) is also described in the work request. Therefore, each work request specifies a set of tasks to be performed, and it is the job of the TASK SET MANAGER to control the execution of that set of tasks.

When a work request arrives, the TASK SET MANAGER parses the work request and initiates construction of the task graph. In XFDPS.1 only a single copy of the task graph is maintained. This copy is stored at the node where the TASK SET MANAGER for the work request resides. At this stage of work request processing, the task graph contains only the initial resource requirements for the work request; i.e., that information obtained from the work request itself.

The next step involves sending a message to the FILE SYSTEM MANAGER residing on the same node as the TASK SET MANAGER requesting file availability information concerning the files needed by this work request. A message is also sent to the PROCESSOR UTILIZATION MANAGER residing on the same node as the TASK SET MANAGER requesting processor utilization information. This includes the latest utilization information that this particular node has obtained from all other nodes.

When the file availability information and processor utilization information arrive, a work distribution and resource allocation decision is made by the TASK SET MANAGER. At this point specific files are chosen from



Distribution of Components:

TASK SET MANAGER - 1 per user terminal or executing command file

FILE SYSTEM MANAGER - 1 per node

FILE SET MANAGER - 1 per node

PROCESSOR UTILIZATION MANAGER - 1 per node

PROCESSOR UTILIZATION MONITOR - 1 per node

PROCESS MANAGER - 1 per node

Figure 26. The XFDPS 1 Model of Control

the list of files found available and specific processors are chosen as sites for the execution of the various tasks of the work request's task set. It is anticipated that the performance of the overall system as well as the individual work requests will be affected by the nature of the resource allocation and work distribution decision, but this topic will not be investigated in this dissertation (see [Chu80, Shar81, Ston78] for work in this area); instead, all experiments use a single strategy in which a process is assigned to execute on the same node that its object code resides. Data files are not moved either but accessed from the node on which they originally resided.

Once the allocation decision is made, a request for the locking of the chosen files is sent by the TASK SET MANAGER to the FILE SYSTEM MANAGER residing on the same node as the TASK SET MANAGER. The desired type of access (READ or WRITE) is also passed with the lock request. Multiple readers are permitted, but readers are denied access to files already locked for writing, and writers are denied access to files locked for reading or writing. If the FILE SYSTEM MANAGER informs the TASK SET MANAGER that all the desired files have been successfully locked, execution of the work request can be initiated. If the locking operation is not successful, the work request is aborted, and the necessary cleanup operations are performed. The next step after successful file allocation is to send a series of messages to the PROCESS MANAGERs on the various nodes that have been chosen to execute the tasks of the task set informing them that they are to execute a specific subset of tasks.

When a task terminates, its PROCESS MANAGER reports back to the TASK SET MANAGER and indicates the reason for the termination (normal or abnormal). When an indication of an abnormal termination is received, the remaining active tasks of the task set are terminated.

After all tasks of a task set have terminated, one of three possible actions occurs. If the source of commands is a user terminal, the user is prompted for a new command. If the source of commands is a command file, the next command is obtained. Finally, if the source is a command file and all the commands have been executed, the TASK SET MANAGER is deactivated and the PROCESS MANAGER on the node where the command file was being executed is informed of the termination of the command file.

4.1.2 File System Manager

Replicated on each node of the system is a component called the FILE SYSTEM MANAGER. This module handles the file system requests from all of the TASK SET MANAGERs including requests for file availability information and requests to lock or release files. FILE SYSTEM MANAGERs do not possess any directory information. Therefore, to locate a file it is necessary that all nodes be queried as to the availability of the file.

The FILE SYSTEM MANAGER satisfies the requests by consulting with the FILE SET MANAGERs located on each node of the system. For example, when the FILE SYSTEM MANAGER receives a request for file availability information, messages are prepared and sent to all FILE SET MANAGERs. The FILE SYSTEM MANAGER collects the responses, and when responses from all FILE SET MANAGERs have been obtained, it reports the results to the TASK SET MANAGER that made the request. Requests for the locking or releasing of files are handled in a similar manner.

4.1.3 File Set Manager

The files residing on each node of the system are managed separately from the files on other nodes by a FILE SET MANAGER that is dedicated to managing that set of files. The duties of the FILE SET MANAGER include providing file availability information to inquiring FILE SYSTEM MANAGERs and reserving, locking, and releasing files as requested by FILE SYSTEM MANAGERs. It should be noted that a side effect of gathering file availability information is the placement of a reservation on a file that is found to be available.

4.1.4 Processor Utilization Manager

Also present on each node is another component of the executive control, the PROCESSOR UTILIZATION MANAGER. This module is assigned the task of collecting and storing processor utilization information, which is obtained from the PROCESSOR UTILIZATION MONITORs residing on each of the nodes. When a TASK SET MANAGER asks the PROCESSOR UTILIZATION MANAGER for utilization information, the PROCESSOR UTILIZATION MANAGER responds with the data available at the time of the query.

4.1.5 Processor Utilization Monitor

Each node of the system also has a PROCESSOR UTILIZATION MONITOR that is responsible for collecting various measurements needed to arrive at a value describing the current utilization of the processor on which the PROCESSOR UTILIZATION MONITOR resides. The processor utilization value is periodically transmitted to the PROCESSOR UTILIZATION MANAGERs on all nodes.

4.1.6 Process Manager

Residing on each node of the system is a PROCESS MANAGER whose function is to supervise the execution of processes executing on the node on which it resides. The PROCESS MANAGER is responsible for activating and deactivating processes. If the execution file for a process is an object file, the PROCESS MANAGER will load the object file into memory. This file may reside either locally or on a distant node. If the execution file is a command file, the PROCESS MANAGER sees that a TASK SET MANAGER is activated to respond to the commands of that command file. The PROCESS MANAGER is also responsible for handling process termination, which involves releasing local resources held by the process and informing the TASK SET MANAGER that requested the execution of the process as to the termination of the process.

4.1.7 File Process

In order to provide file access in a manner that is uniform with the operation of the rest of the system, another type of control process called a FILE PROCESS is utilized. An instance of a FILE PROCESS is created for each user of a file. Therefore, if process 'A' is accessing file 'X' and process 'B' is also accessing file 'X', there will be two instances of a FILE PROCESS, each responsible for a particular access to file 'X'. Communication between FILE PROCESSes and user processes (file reads and writes) or between FILE PROCESSes and PROCESS MANAGERs (loading of object programs) is handled in the same manner as communication between user processes.

4.2 The XFDPS.2 Control Model

The XFDPS.2 model of control differs from the XFDPS.1 model in the manner in which file management is conducted. In this model a centralized directory is maintained. In the Appendix the component named FILE SYSTEM MANAGER maintains this directory. This component resides on only one node, the node where the file system directory is maintained. TASK SET MANAGERs communicate

directly with this component in order to gain availability information, lock files, or release files.

When a file is locked, it is necessary to create a FILE PROCESS in order to provide access to the file. To accomplish this task, the FILE SYSTEM MANAGER sends a message to the node where the file resides requesting activation of a FILE PROCESS providing access to the file. Once this process is created, the FILE SYSTEM MANAGER is given the name of the FILE PROCESS which it then returns to the TASK SET MANAGER that requested the file lock.

4.3 The XFDPS.3 Control Model

In the XFDPS.1 model of control a search for file availability information encompassing all nodes is conducted for each work request. Obtaining this global information is important when one is attempting to obtain optimal resource allocations. In those instances where this is not important, a slight variation on the search strategy may be utilized. This strategy is the distinguishing feature of the XFDPS.3 model of control.

Instead of immediately embarking on a global search, a search of local resources (i.e., resources that reside on the same node where the work request originated) is conducted. If all of the required resources are located, no further searches are conducted, and the operations of locking files, activating process, etc., described for model XFDPS.1, are executed. If on the other hand all required resources could not be found, the strategy of model XFDPS.1 is utilized.

4.4 The XFDPS.4 Control Model

The XFDPS.4 model of control utilizes a file management strategy similar to that of the ARAMIS Distributed Computer System in which multiple redundant file system directories are maintained on all nodes of the system. (Since detailed information about the system described in [Caba79a,b] is not available, model XFDPS.4 cannot be claimed to be an accurate model of the ARAMIS system.)

To preserve the consistency of the redundant copies of the file system directory and to provide mutually exclusive access to resources, the following steps are taken. A control message, the control vector (CV), is passed from

node to node according to a predetermined ordering of the nodes. The holder of the CV can either release, reserve, or lock files. Therefore, each node collects file system requests and waits for the CV to arrive. Once in possession of the CV, a node can perform the actions necessary to fulfill the requests it has collected.

The modifications to the file system directory are then placed into a message called the update vector (UPV) which is passed to all nodes in order to bring all copies of the file system directory into a consistent state. When the UPV returns to the node holding the CV, all updates have been recorded, and the CV can be sent on to the next node.

4.5 The XFDPS.5 Control Model

In the XFDPS.5 model, files are not reserved when the initial availability request is made, and they are locked only after the work distribution and resource allocation decision has been made. This strategy leads to the possibility of generating an allocation plan that is impossible to carry out if a file chosen for allocation has been given to another process during the interval in which the resource allocation decision is made. In the previous models the executive control is assured of an allocation being accepted, assuming no component fails.

4.6 The XFDPS.6 Control Model

In the XFDPS.1 model the task graph for a particular work request is maintained as a single unit and stored on only one node, the node at which the work request originates. The XFDPS.6 model of control utilizes a slightly different strategy. The task graph is constructed on a single node, but once a work distribution and resource allocation decision has been made, portions of the task graph are sent to various nodes. Specifically, those nodes chosen to execute the various tasks of the task graph are given that portion of the task graph for which they are responsible. Each node must activate the tasks assigned to it and collect termination information concerning those tasks. When all tasks assigned to a particular node have terminated, the node where the work request originally arrived is informed of their termination. One can view this strategy as a two-level hierarchy.

SECTION 5

THE METHOD OF PERFORMANCE ANALYSIS

In order to obtain quantitative information concerning the relative performance of the various models of control, simulation experiments were conducted. The goals of these experiments were to validate the models of control described in Chapter IV and gather data on their relative performance. In order to be able to express the differences between the various models, it was necessary that the simulator provide for the specification of relatively low level features of the control models.

The goals described above necessitate the establishment of several requirements for the simulator. In order to handle low level control problems and document solutions to these problems, the control models must be defined in a language capable of clearly expressing the level of detail required at this stage of design. Because a number of models are to be tested, it is important that the coding effort required to describe these different models be minimized.

It is expected that the architecture of the network as well as that of individual nodes in the network will affect the relative performance of various control models. Therefore, it is also important to be able to easily modify various architectural attributes. This includes network connectivity, network link capacities, and the capacities and processing speeds of the individual nodes of the network.

Validation of control models is one of the primary goals of the simulation studies. To achieve this goal the simulator must provide the ability to establish specific system states. In other words, specific detailed instances of work requests need to be constructed along with the establishment of specific resource states (e.g., one must be able to set up a series of files in specific locations). These capabilities allow one to exercise specific features of the control models.

The simulation studies also provide performance information. The simulator must utilize a technique for generating work requests reflecting specific distributions. It also needs to collect a variety of performance measurements and generate appropriate statistical results.

5.1 The Simulator

Existing simulators such as the Distributed System Simulator [DuBo81] do not provide the necessary facilities to support the study of executive control characteristics as is required for this work. Therefore, it was necessary to construct an origninal simulator which would provide the experimenter with the ability to examine the behavior of different control models. The simulator is event based and programmed in Pascal. (The programming language Pascal was selected over other languages designed for simulation work because of its availability.) The simulator simulates the hardware components of an FDPS, functions typically provided by local operating systems, functions provided by a distributed and decentralized control, and the load placed upon the system by users attached to the system through terminals.

5.1.1 Architecture Simulated

The hardware organization that is simulated for each node is depicted in Figure 27. The complete system consists of a number of nodes connected by half-duplex communication links. Full-duplex links are simulated by two half-duplex links. Each node contains a CPU, a communications controller, and, perhaps, a number of disks. Connected to each node are a number of user terminals. The disk simulation is such that no actual information is stored; only the delays experienced in performing disk input/output are considered. Also, user interprocess communication (IPC) is simulated with time delays but no exchange of real data takes place. However, IPC between components of the executive control involves both simulation of the time delays involved in message transfer and the actual transfer of control information to another simulated node.

5.1.2 Local Operating System

Components typically found in local operating systems are also simulated. These include the dispatcher and the device drivers. The local operating systems are multitasking systems with each node capable of utilizing a different time slice. User processes are serviced in a first come first served manner and can be interrupted for any of the following reasons: 1) a control process needs to execute (user process is delayed until the control process releases the processor), 2) the user process exhausts its time slice (user process is placed at the end of the READY QUEUE), 3) the user process

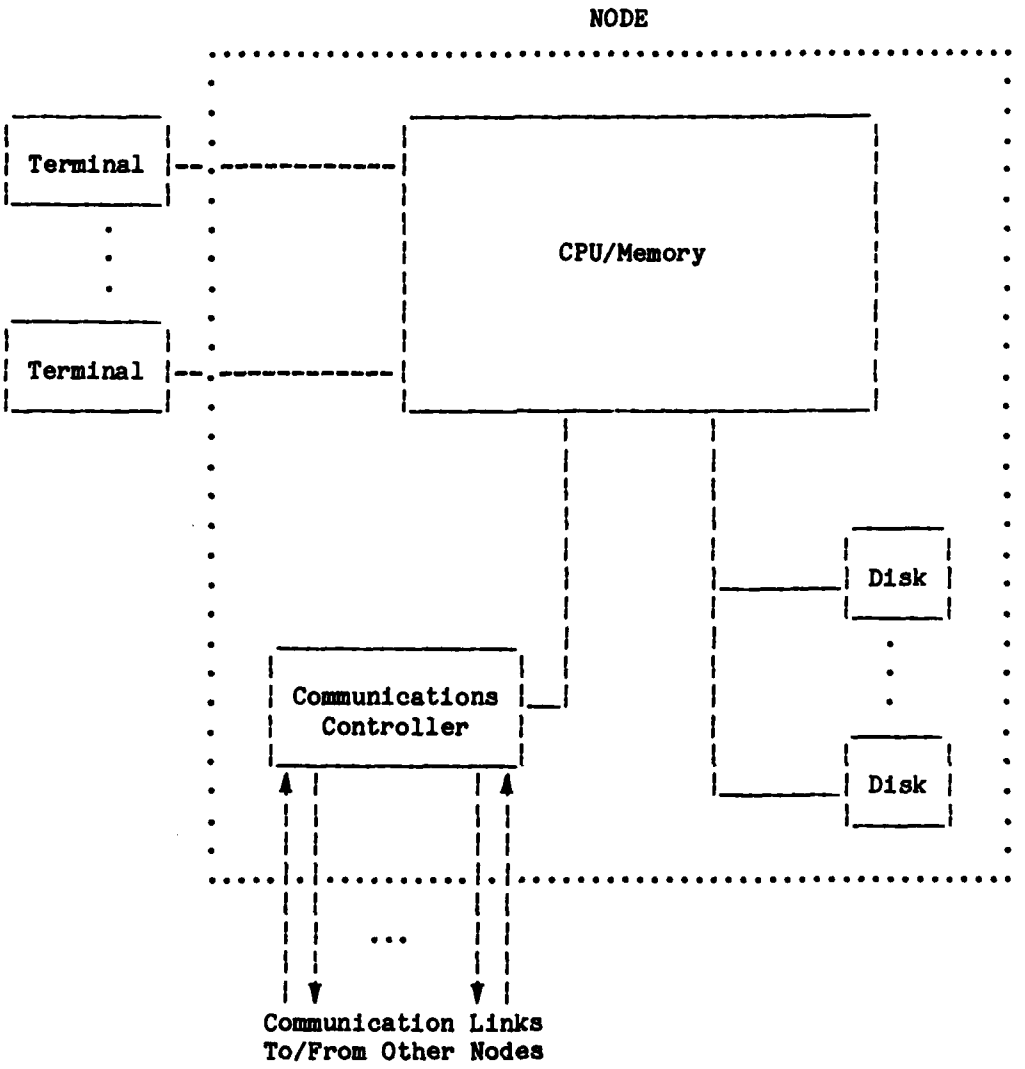


Figure 27. The Architecture Supported by the Simulator for Each Node

attempts to send or receive a message (user process is placed on the MESSAGE BLOCKED QUEUE), or 4) the user process terminates.

The processes serviced by the simulator are capable of performing the following actions: compute, send a message, receive a message, or terminate. A process can access a file by communicating with a FILE PROCESS which is activated for the specific purpose of providing access to the file for this

process. FILE PROCESSes are the only processes that initiate any disk activity. As far as a user process is concerned, a file access functions just like a communication with another process.

The following process queues are maintained: READY QUEUE, DISK WAITING QUEUE, and MESSAGE BLOCKED QUEUE. (See Figure 28.) A newly activated process is placed in the READY QUEUE. The DISPATCHER selects a process from the READY QUEUE to run on the CPU. If the running process exhausts its time slice, it is returned to the READY QUEUE. If it attempts to either send or receive a message, it is placed in the MESSAGE BLOCKED QUEUE where it remains until either the message is placed in the proper link queue (send operation) or a message is received (receive operation). After leaving the MESSAGE BLOCKED QUEUE, a process returns to the READY QUEUE.

The only processes capable of performing disk input/output on the simulator are FILE PROCESSes. These are executive control processes that are assigned to provide access to the files of the file system. When a file process attempts a disk access, it is blocked and placed in the DISK WAITING QUEUE for processes waiting to access that same disk. As the disk requests are satisfied, these processes are returned to the READY QUEUE.

5.1.3 Message System

The communication system consists of a series of half-duplex connections between pairs of nodes. Messages are transmitted using a store-and-forward method. Messages received at intermediate nodes in a path are stored and forwarded to the next node at a time dictated by the communication policy being utilized. For example, the policy may require that the new message be placed at the end of the queue of all messages to be transmitted on a particular link. (This is the policy utilized in all experiments described in this dissertation.)

The message queues available on each node are depicted in Figure 29. If a newly created message is an intranode message, it is placed in the MESSAGE QUEUE; otherwise, it is placed in the LINK QUEUE that corresponds to the communication link over which the message is to be transmitted. Messages are removed from the LINK QUEUEs and transmitted as the communication links become available.

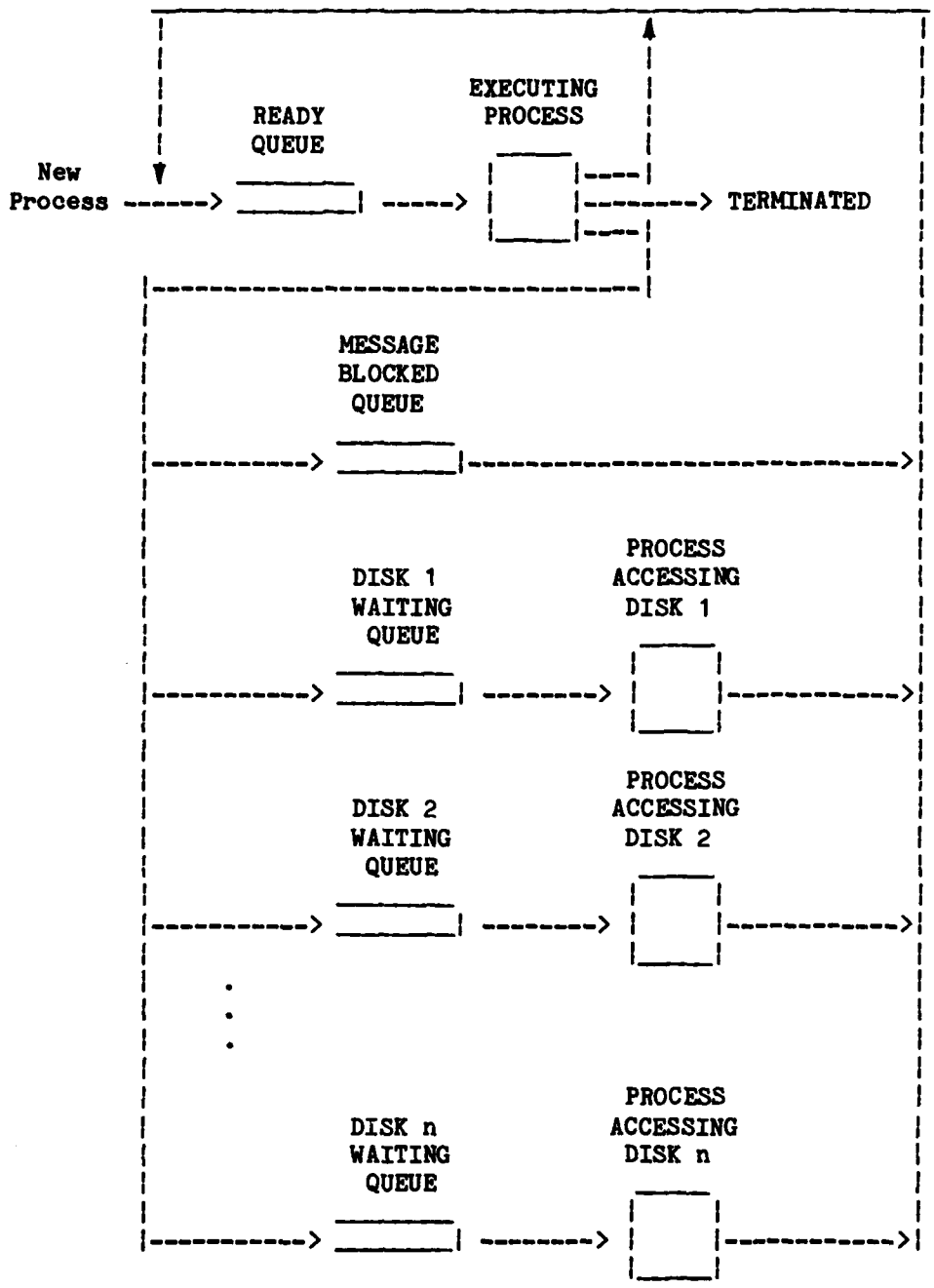


Figure 28. Process Queues on Each Node

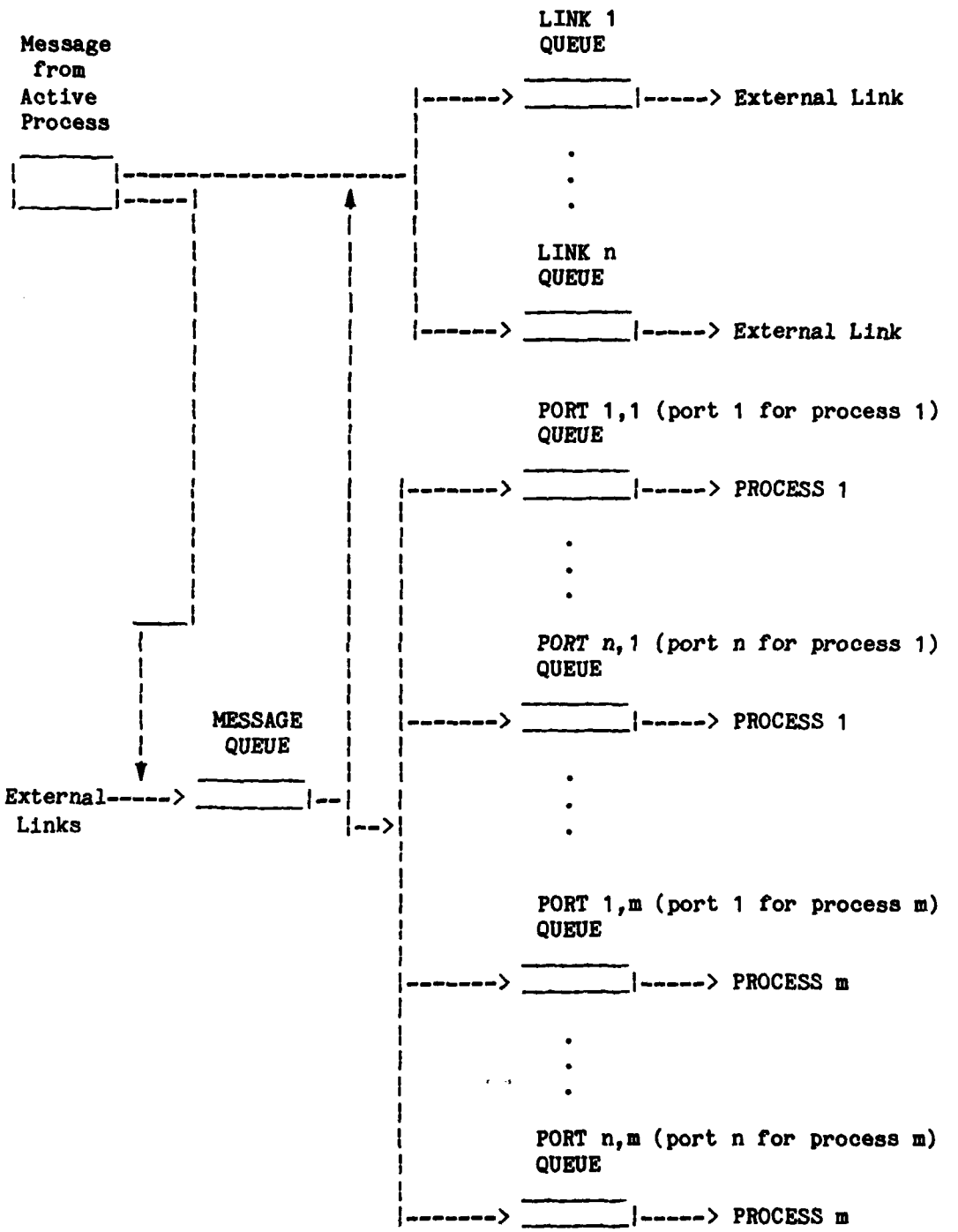


Figure 29. Message Queues on Each Node

Messages in the MESSAGE QUEUE originate either from processes sending intranode messages or from the communication links connected to the node. Messages destined for processes on the same node as the MESSAGE QUEUE are placed in the appropriate PORT QUEUE of the process to which they are addressed. Messages that have not yet reached their destination node are placed in the LINK QUEUE corresponding to the communication link over which the message is to be transmitted.

5.1.4 Input for the Simulator

The simulator requires the following six types of input:

1. Control model
2. Network configuration (i.e., nodes and their connectivity)
3. Work requests
4. Command files
5. Object files
6. Data files

The nature of these inputs and how they are provided to the simulator is described below.

5.1.4.1 Control Model

There are two possible approaches for representing the control model in the simulator: 1) data to be interpreted by the simulator and 2) code that is actually part of the simulator. The first technique requires that the simulator contain a rather sophisticated interpreter in order to provide a convenient language with which one can express a control model that addresses the control problems to a sufficiently low level of detail. The second technique requires the careful construction of the simulator such that those portions of the simulator that express the control model are easily identified and can be removed and modified with minimal effort. No matter how well the portion of the simulator representing the executive control is isolated, it is anticipated that a certain degree of difficulty will be experienced by a new experimenter attempting to investigate new control models. The second technique also requires a recompilation of the simulator code each time a control model modification is performed.

The problems involved in constructing a sophisticated interpreter are much greater than those faced in organizing the simulator so that the portions

of code expressing the control model are easily isolated. Therefore, in this simulator, the control models are expressed in Pascal and are actually part of the simulator rather than being separate input to the simulator.

5.1.4.2 Network Configuration

The attributes provided as input to the simulator which are concerned with the physical configuration of the FDPS are provided in Table 2. Figure 30 describes the syntax of the statements used to enter the FDPS configuration information. Two types of input can be provided, node configuration information and communication linkage information. Each statement beginning with the letter 'n' describes the configuration of the node which is identified by the digit following the 'n'. This statement describes certain characteristics concerning the processor at the node (memory capacity, processing speed, and the length of a user time slice) and the peripheral devices (user terminals and disks) attached to the processor. Each statement beginning with the letter 'l' describes a half-duplex communication link between two nodes. It identifies the source and destination nodes by their identification number (the digit following the letter 'n' on statements describing nodes) and indicates the effective bandwidth of the communication link. It is assumed that all messages are transmitted at this speed, and no attempt is made to simulate errors in transmission and the resulting retransmissions.

5.1.4.3 Work Requests

Work requests are assumed to originate from two sources: 1) directly from a user, or 2) through command files. The syntax of a work request is given in Figure 31. This syntax is a subset of the command language available through the Advanced Command Interpreter of the Georgia Tech Software Tools System [Akin80] (see Figure 1).

In order to simulate the load generated by users entering work requests from user terminals, a population of work requests is created. The form of the input for creating the work request population is provided in Figure 32. Each line of input contains a series of node identifiers followed by a colon which is followed by a work request. The node identifiers indicate which nodes are to contain the given work request as a member of the node's population of work requests. Therefore, the result of this input is the construction of a population of work requests for each node. In a subsequent

Table 2. Physical Configuration Input to the Simulator

Node Information

Memory Capacity (bytes)
Processing Speed (Instructions/sec)
Size of a Time Slice (microseconds)
Number of Attached User Terminals
Number of Attached Disks
Disk Transfer Speed (bytes/second)
Average Disk Latency (microseconds)

Link Information

Identities of the Source and Destination Nodes
Bandwidth (bytes/second)

paragraph, the nature of the load generator is discussed and indicates how this information is utilized.

5.1.4.4 Command Files

Command files are *constructed* for the simulator using the syntax described in Figure 33. This input specifies a unique name for the file, the simulated node at which the file resides, and the commands contained in the file. These commands conform to the syntax of work requests presented in Figure 31. These statements provide one with the ability to construct command files on particular nodes which are referenced either by commands originating from user terminals or other command files.

5.1.4.5 Object Files

Figure 34 depicts the syntax used to express object files in the simulator. The input specifies a unique name for the file, the simulated node at which the file resides, the length of the file in bytes, and the simulation script. The script contains a series of statements that describe the process actions that are to be simulated. There are five actions which can be simulated: 1) compute, 2) receive a message, 3) send a message, 4) loop back to a previous command a specific number of times, and 5) terminate the process simulation. By appropriately combining these commands, one can construct a script which simulates the activities of a given user process.

```
<entry> ::= <link> | <node>

<link> ::= l <from> <to> <bandwidth>  (all links are half-duplex)

<node> ::= n <node id> <memory> <speed> <timeslice> <terminals>
           <disk> <disk speed> <disk latency>

<from> ::= <node id>

<to> ::= <node id>

<node id> ::= <integer>

<bandwidth> ::= <integer (link bandwidth in bytes per second)>

<memory> ::= <integer (main memory in bytes)>

<speed> ::= <integer (average speed of the CPU in instructions per
            second)>

<timeslice> ::= <integer (microseconds)>

<terminals> ::= <integer (number of attached user terminals)>

<disk> ::= <integer (number of attached disks)>

<disk speed> ::= <integer (transfer speed of disk in bytes/sec)>

<disk latency> ::= <integer (average disk latency in microseconds)>

<integer> ::= <digit> { <digit> }


Examples:

   n 1 256000 5000000 1000 50 3 500000 100
      (Node #1 has 250K bytes of memory, processes at the rate of
       5 MIPS, has a time slice of 1000 microseconds, has 50 user
       terminals attached to it, has 3 disks attached to it,
       each disk can transfer at the rate of 500,000 bytes/sec,
       and each disk has an average latency of 100 microseconds.)

   l 5 6 4000000
      (This link connects node 5 to node 6 with a half-duplex
       communication path that can transmit at the rate of
       4 million bytes/sec.)
```

Figure 30. Syntax of FDPS Configuration Input for the Simulator

```
<work request> ::= <logical net>

<logical net> ::= <logical node> { <node separator>
                  { <node separator> } <logical node> }

<node separator> ::= , | <pipe connection>

<pipe connection> ::= [ <port> ] '|' [ <logical node number> ]
                      [ .<port> ]

<port> ::= <integer>

<logical node number> ::= <integer> | $ | <label>

<logical node> ::= [ :<label> ] <simple node>

<simple node> ::= { <i/o redirector> } <command name>
                  { <i/o redirector> }

<i/o redirector> ::= <file name> '>' [ <port> ]  |
                     [ <port> ] '>' <file name>  |
                     [ <port> ] '>>' <file name> |
                                '>>' [ <port> ]

<command name> ::= <command file name> | <object file name>

<label> ::= <identifier>

<file name> ::= <data file name>

<identifier> ::= <letter> { <letter> | <digit> }

<integer> ::= <digit> { <digit> }
```

Figure 31. Work Request Syntax
(Based on [AKIN80])

5.1.4.6 Data Files

Data file descriptions, depicted in Figure 35, are the final type of input information which can be presented to the simulator. The data file input contains an identifying name, a node identification indicating the file's simulated location, and a specification of the file size. Data is not actually stored by the simulator.

```
<work request population> ::= <work request entry>
                                        .
                                        .
                                        .
                              <work request entry>

<work request entry> ::= { <node identifier> } : <work request>

<node identifier> ::= <integer>

<work request> ::= (see Figure 31)

<integer> ::= <digit> { <digit> }


Examples:

   1 2 3 4 5 : pgm1 | pgm2       { the work request 'pgm1 | pgm2'
                                   is available on nodes 1, 2, 3,
                                   4, and 5 }
   1 3 : pgm1                    { the work request 'pgm1' is
                                   available on nodes 1 and 3 }
```

Figure 32. Syntax of Work Request Population Input to the Simulator

5.1.5 The Simulator Design

The simulator is composed of several modules (see Table 3). In each module, closely related data structures and the procedures that modify these data structures are defined. The only access to the data structure is through these procedures. This design allows one to isolate the portion of the simulator that represents the model of control and conduct experiments with various perturbations of the control model. Without this type of design, each perturbation could easily require significant changes to the entire simulator. The simulator is composed of the following modules: a node module, message system module, file system module, command interpreter module, task set and process manager module, and a load generator module. The bulk of code representing the simulated executive control is contained in the FILE SYSTEM and TASK SET AND PROCESS MANAGER modules.

```
<command file> ::= C <node id> <command file name>
                   { <work request> }
                   ENDC

<node id> ::= <integer>

<command file name> ::= <up to 8 characters>

<work request> ::= (see Figure 31)

<integer> ::= <digit> { <digit> }


Examples:

  C 1 cfile1
  pgm1 | pgm2 1|a 2|b :a pgm3 | pgm4 |c.1 :b pgm5 | pgm6 |.2 :c pgm7
  pgm1 | pgm5
  ENDC
```

Figure 33. Syntax of Command File Descriptions for the Simulator

5.1.5.1 Node Module

The NODE MODULE simulates the hardware activities of each node (e.g., the processor and attached disks). This includes the simulation of user activities as specified by process scripts and the simulation of disk traffic. In addition, this module provides the local operating system functions of dispatching, blocking processes for message transmission or reception, and unblocking processes.

5.1.5.2 Message System

All activities dealing with messages are handled by the MESSAGE SYSTEM. Among the services provided by this module are the following: 1) routing of messages, 2) placement of messages in LINK QUEUEs, 3) transmission of messages across a link, 4) transmission of acknowledgement signals to the source end of a link, and 5) placement of messages in PORT QUEUEs.

5.1.5.3 File System

The FILE SYSTEM stores the various types of files, which include object, command, and data files. It stores the scripts for object files and provides access to the scripts. Similarly for command files, it stores the work

```
<object file> ::= 0 <node id> <file name> <file length>
                  { <action> }
                  ENDO

<node id> ::= <integer>

<object file name> ::= <up to 8 characters>

<object file length> ::= <integer>

<action> ::= <comp> | <loop> | <rcv> | <send> |  <term>

<comp> ::= c <# of instructions>

<loop> ::= l <instruction #> <count>

<rcv> ::= r <port>

<send> ::= s <port> <size (bytes)>

<term> ::= t

<# of instructions>, <instruction #>, <count>, <port>,
   <size> ::= <integer>

<integer> ::= <digit> { <digit> }


Examples:

   0 1 object1 1000  (object file is 1000 bytes long)
   c 25              (simulate 25 computation instructions)
   l 1 10            (loop back to the first instruction 10 times)
   r 2               (read a message from port 2)
   s 4 100           (send a 100 byte long message to port 4)
   t                 (terminate the execution of this process)
   ENDO
```

Figure 34. Syntax of Object File Descriptions for the Simulator

requests for each command file and controls access to the file. It is in this module that the file management strategy for each model of control is simulated. The reader is referred to Chapter IV for a description of each control model including specific details concerning the file management strategies that are simulated.

```
<data file> ::= D <node id> <data file name> <size>

<node id> ::= <integer>

<data file name> ::= <up to 8 characters>

<size> ::= <integer (bytes)>

<integer> ::= <digit> { <digit> }


Examples:

   D 3 testfile 100000      (defines a data file named 'testfile'
                             which will reside on node 3 and will
                             contain 100,000 bytes of information)
```

Figure 35. Syntax of Data File Descriptions for the Simulator

5.1.5.4 Command Interpreter

The COMMAND INTERPRETER parses work requests and constructs the task graph describing the initial resource requirements for a work request.

5.1.5.5 Task Set and Process Manager

The TASK SET AND PROCESS MANAGER performs all control activities required to manage all phases of execution of a work request. This includes activating the COMMAND INTERPRETER; communicating with the FILE SYSTEM in order to gather information, allocate files, or deallocate files; performing work distribution and resource allocation; and managing active processes.

5.1.5.6 Load Generator

Work request traffic originating from the user terminals attached to each node is simulated by the LOAD GENERATOR. A series of work requests provided by a user at a terminal is called a user session. To simulate a user session, the LOAD GENERATOR randomly chooses a session length from an interval specified by the experimenter. A session starting time (measured in seconds) is also chosen at random from an interval specified by the experimenter. Each work request for the user session is chosen at random from the population of work requests originally created for each node via the input statements described above (see Figure 32). The LOAD GENERATOR also simulates the "think time"

AD-A113 624 GEORGIA INST OF TECH ATLANTA SCHOOL OF INFORMATION A--ETC F/G 9/2
DISTRIBUTED AND DECENTRALIZED CONTROL IN FULLY DISTRIBUTED PROC--ETC(U)
DEC 81 T G SAPONAS N00014-79-C-0873
UNCLASSIFIED GIT-ICS-81/18 NL

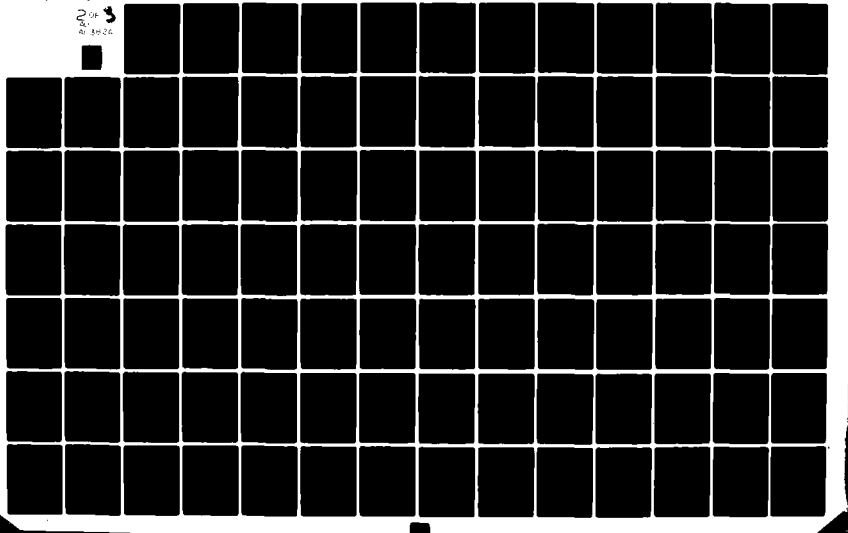


Table 3. Simulator Modules

- Node Module
  - Process user scripts
  - Simulate disk activity
  - Manage local processes
- Message System Module
  - Route messages
  - Message management at each node
  - Simulate the transmission of messages
- File System Module
  - Perform typical file management operations
    - Locate files
    - Provide access control to files
  - Store and retreive data for object files and command files
- Command Interpreter Module
  - Parse command lines and return task graphs
- Task Set and Process Manager Module
  - Task set management
    - Contact file system for file availability information
    - Formulate work distribution and resource allocation decision
    - Contact file system for file allocation
    - Contact process manager to activate processes
    - Inform user of work request completion
  - Process management
    - Load processes
    - Detect process termination and inform task set manager
- Load Generator Module
  - Simulate user activity

between work requests by randomly choosing a time (measured in seconds) from another interval specified by the experimenter.

5.1.6 Performance Measurements

Performance measurements covering the following three types of data are made: 1) the quantity of message traffic, 2) the magnitudes of various queue lengths and their associated waiting times, and 3) the size of average work request response times and throughput.

To identify the impact of the executive control on the communication system, various communication measurements are obtained. A cumulative total of the number of user messages and control messages over the entire system is maintained. This allows one to compare the number of control messages to the number of user messages and thus identify how the communication system is being utilized. In addition, a count, again categorized by user messages and control messages, is maintained in matrix form to identify the total number of messages originating at a particular node and destined for every other node. Traffic counts on each communication link are also recorded according to their classification as user messages or control messages. Finally, activity in the LINK QUEUEs, where messages wait to be transmitted over each link, is recorded. All of these measurements include minimum queue length, maximum queue length, average queue length, minimum waiting time in the queue, maximum waiting time, and average waiting time.

In addition to measurements concerned with the LINK QUEUEs, a similar analysis of process queues is performed. The queues on each node that are analyzed are the READY QUEUE (processes waiting for access to the CPU), MESSAGE BLOCKED QUEUE (processes that are either waiting to place a message in a LINK QUEUE or processes waiting to receive a message), and DISK WAITING QUEUEs (processes waiting for access to a particular disk). The types of measurements obtained are identical to those for the LINK QUEUEs.

To identify the effectiveness of the control strategy, measurements are obtained that identify how effectively user processing is accomplished. For each node and cumulatively for all nodes, the following measurements are obtained for user sessions, work requests, and processes:

1. The total number of user sessions, work requests, and processes.
2. The average service time for each user session, work request, and process.
3. The average response time for each user session, work request, and process.
4. The throughput for user sessions, work requests, and processes.

5.2 The Simulation Environment

Four groups of performance experiments were conducted in this research project. The first group of experiments observed the behavior of a system in which only control message traffic was present on the communication system. The second group of experiments introduced user message traffic. The third group of experiments was similar to the first group in that only control message traffic was present on the communication system, but a different type of work request was utilized. The work requests that were processed in the first two groups of experiments required significant processing time to perform the actions specified in the request. The work requests utilized in the third group of experiments represented jobs requiring only a small quantity of computation. Work requests were selected from a mixed population of two different types of work requests in the fourth group of experiments. The two types of work requests corresponded to those used in the second and third groups of experiments respectively.

The environment in all experiments consisted of a network of five nodes interconnected in various ways providing five different interconnection topologies: 1) a unidirectional ring, 2) a bidirectional ring, 3) a star, 4) a fully connected network, and 5) a tree. (See Figure 36.) The nodes of each network (see Figure 27) were all homogeneous, and each consisted of a processor capable of executing one million instructions per second. Connected to each node were ten user terminals and three disk drives. The disks were assumed to be identical, each with an average latency of 100 microseconds and a transfer rate of 500,000 bytes per second.

5.2.1 Environmental Variables

In addition to different topologies, the bandwidth of the communication links and the model of control were also varied for the experiments. Table 4 provides a brief comparison of the various models. Only the first four models of control (XFDPS.1, XFDPS.2, XFDPS.3, and XFDPS.4) were utilized in these initial experiments. Models XFDPS.5 and XFDPS.6 differ from model XFDPS.1 in details that were not examined in these experiments. Therefore, they were not included in the simulation studies because their observable results would have been identical to those of XFDPS.1. Models XFDPS.5 and XFDPS.6 demonstrate that significant variations in design may not necessarily result in per-

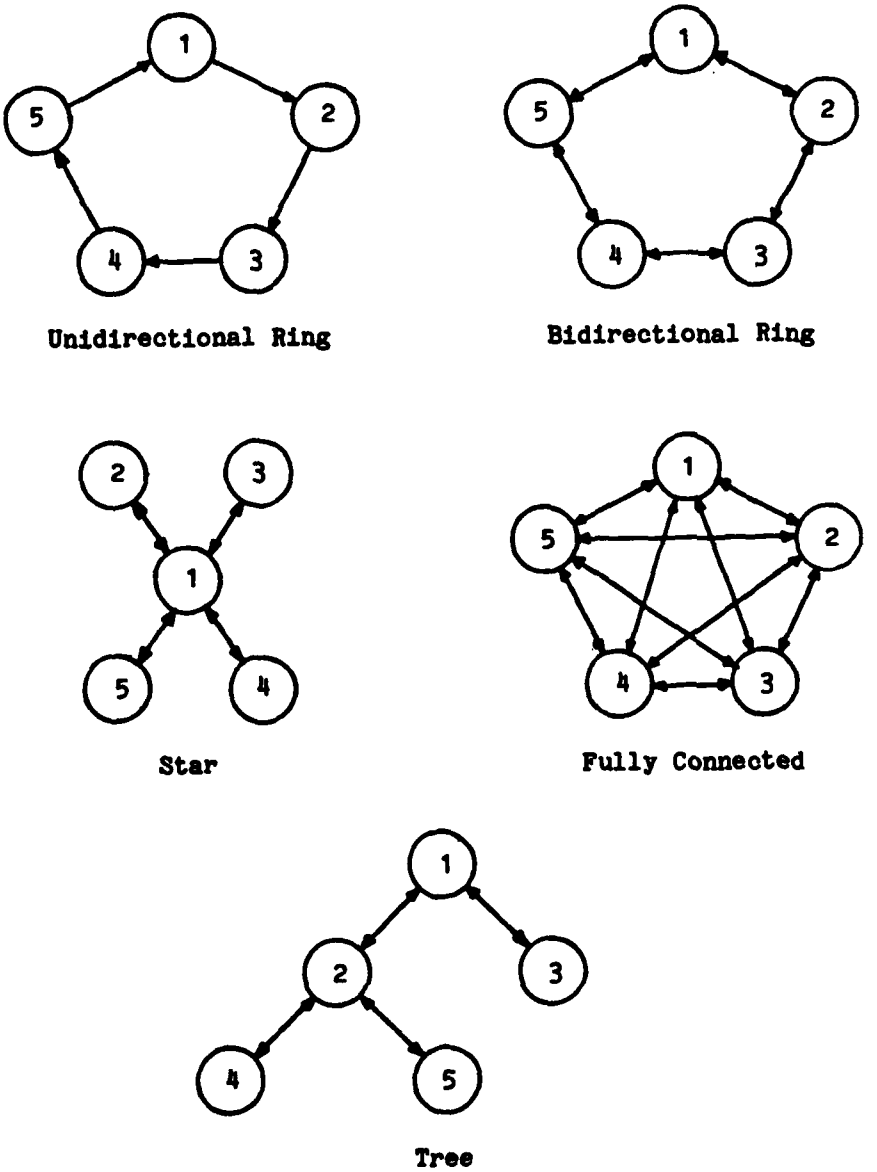


Figure 36. Network Topologies

formance differences under all circumstances. Finally, it should be noted that the central directory of model XFDPS.2 is maintained on node 1 in all experiments.

5.2.2 Environmental Constants

Several environmental features were held constant for all experiments. In all cases, it was assumed that all control messages were 50 bytes long. All control models utilized the same policy for distributing work and allocating resources. This policy simply required all processes to execute on the node where the object code for the process resided. There was only one copy of the object code for each process in the network for these initial experiments. The work distribution and resource allocation policy utilized for these tests required that data files be accessed at the location where they originally resided and not be moved prior to execution. In every experiment all files were unique, thus leaving the control with only one resource allocation alternative.

In the first two groups of experiments, the work requests arriving at all nodes were of the type 'in> cmnd'. The data file 'in' provided input to the process resulting from the loading of the object file 'cmnd'. This provided an environment in which files were accessed only by means of reads thus eliminating the possibility that certain work requests were either delayed or aborted due to insufficient resources. Therefore, it was guaranteed that all control activity resulted in the successful completion of a work request.

In the first group of experiments, the object file 'cmnd' and data file 'in' were located on the same node. This meant that all file accesses were local file accesses; and, thus, control message traffic was free of competition by user messages for communication resources. This provided an environment in which the effects of the control models could be observed without the influence of an unpredictable collection of user messages.

In the second group of experiments, the object file 'cmnd' and data file 'in' were located on different nodes. File 'cmnd' was located on node i and file 'in' was located on node j where

Table 4. Comparison of the Models of Control

Model	File System Directory	Technique for Gathering Availability Information	Time When Files are Reserved or Locked	How is the Task Graph Maintained
1	partitioned and distributed	query all nodes	before resource allocation and work distribution decision	single structure on node where work request arrived
2	single centralized copy	query the central node	before resource allocation and work distribution decision	single structure on node where work request arrived
3	partitioned and distributed	first query locally and then query globally if necessary	before resource allocation and work distribution decision	single structure on node where work request arrived
4	identical copies replicated on all nodes	all queries are delayed until control vector arrives	before resource allocation and work distribution decision	single structure on node where work request arrived
5	partitioned and distributed	query all nodes	after resource allocation and work distribution decision	single structure on node where work request arrived
6	partitioned and distributed	query all nodes	before resource allocation and work distribution decision	multiple subgraphs on the nodes involved in the execution of the tasks

$$j = \begin{cases} i + 1, & \text{if } i < 5 \\ 1, & \text{if } i = 5 \end{cases}$$

This meant each file access required transmission of data on the communication system. These experiments were designed to demonstrate what happens to the performance of the control models when additional traffic is present on the communication system.

The object files in each case specified the execution of the same script, which is depicted in Figure 37. This script describes a process that alternately computes and reads from a data file for 501 iterations. Given the speed of the processors utilized in the experiments, this results in a CPU utilization of approximately five seconds for each process.

```
c  10000      { 10,000 compute instructions }
r  1          { read from port 1 }
l  1  500     { loop back to instruction one 500 times }
t             { terminate the process }
```

Figure 37. The Script Utilized by all Processes in Group 1 and 2 Experiments

In the third group of experiments, the work requests arriving at all nodes were of the form 'cmnd'. This simply specified the execution of an object file which required no input file and produced no output file. Figure 38 depicts the script that represents the actions of the object file named in each request. The actions of the script specify only a short computation resulting in a CPU utilization of approximately 0.01 seconds, given the assumed speed of the processors in these experiments.

A population composed of two different types of work requests corresponding to those utilized in group 2 and group 3 experiments, respectively, were used in the fourth group of experiments. The location of object-data file pairs for one type of work requests and object files for the other type were

```
c  10000      { 10,000 compute instructions }
t               { terminate the process }
```

Figure 38. The Script Utilized by all Processes in Group 3 Experiments

identical to that described for the group 2 and group 3 experiments.

As mentioned in the discussion of the LOAD GENERATOR, the experimenter must provide several intervals from which random values are selected as input to the simulator. In Table 5 the values utilized in the experiments are provided. User sessions can possess from one to one hundred work requests. The first user session for each terminal is begun at some time between one simulated second and fifteen simulated seconds. The delay between the completion of one user session and the start of a new one on the same terminal also ranges from one to fifteen simulated seconds. Similarly, the delay between work requests of a user session (user "think time") ranges from one to fifteen seconds. Identical intervals are utilized for the delay between user sessions and the delay between work requests because only statistics concerning work requests are utilized in this study. Statistics concerning user sessions are not considered important.

In order to observe steady state behavior, the start of statistics gathering was delayed until the simulation proceeded for some time. In these studies, statistics were gathered from 30 until 330 simulated seconds. The computer time required to perform the magnitude of calculations involved in a simulation experiment was the factor limiting the length of time that was simulated. The value of thirty seconds for the start of statistics gathering seemed satisfactory because it provided enough time for all terminals to have generated one work request (each terminal must supply its first work request by fifteen simulated seconds) and, in some cases, have all or at least a substantial portion of the computation for the first work request completed with additional work requests also active.

In studies such as these, it is desirable to provide an identical load for all simulation experiments, but the nature of the system under examination makes this impossible. To provide an identical load, one would have to

Table 5. Values of User Specified Intervals

Variable	Interval
User Session Length	[1, 100] work requests
User Session Starting Time	[1, 15] seconds
Delay Between User Sessions	[1, 15] seconds
Delay Between Work Requests	[1, 15] seconds
Statistics Gathering Interval	[30, 330]* seconds [30, 630]** seconds

* used in all group 1 and group 3 experiments and all group 2 experiments except those using a unidirectional ring with a bandwidth of 50,000 bytes/sec

** used in all group 4 experiments and in group 2 experiments using a unidirectional ring with a bandwidth of 50,000 bytes/sec

guarantee that the work requests are presented to the simulator in the same order and at the same exact time intervals for each experiment. The control models, though, are composed of autonomous components and by their design will process work requests asynchronously on each node at different rates. This implies that even if the work requests at each node are presented in the same order, the load provided to the simulator will be different because the timing of work request arrivals may vary.

To clarify this point, consider the following example. Assume the loads provided to nodes 1 and 2 are as shown in Figure 39. This figure depicts the order in which the work requests arrive at each node. Because the control models process work requests at different rates, different processing sequences are obtained for the control models. Figure 40 depicts the sequence for model 1 and Figure 41 depicts that for model 2. Thus, although the loads at each node are controlled, it is impossible to control the sequence of work requests on all nodes collectively.

Since identical loads cannot be provided, an attempt is made to construct an unbiased load. This is the task of the LOAD GENERATOR. Its

design utilizes random selection for work requests and delays between work requests in order to create an unbiased environment.

Load at Node 1	Load at Node 2
WR1	WR5
WR2	WR6
WR3	WR7
WR4	WR8

Figure 39. Example of Loads Presented to Two Nodes

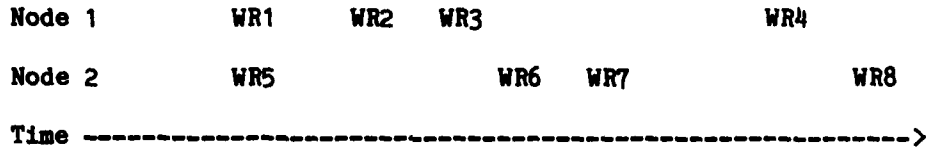


Figure 40. Sequence of Work Request Arrivals When Using Model 1

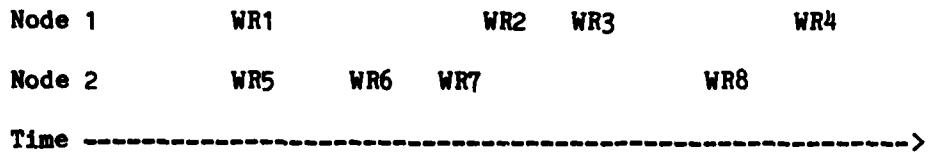


Figure 41. Sequence of Work Request Arrivals When Using Model 2

SECTION 6

SIMULATION RESULTS

6.1 Work Requests Utilizing Only Local File Access

6.1.1 The Environment

This group of experiments was designed to examine the performance of the control models in an environment in which only control message traffic was present on the communication links. Each work request in the pool utilized by the LOAD GENERATOR specified an object-data file pair in which both the object file and data file resided on the same node. Thus, file accesses by processes did not use the communication system. There were equal numbers of object-data file pairs on each node. The probability that a newly arriving work request named an object-data file pair residing on node i was 1/5 for i = 1 to 5.

In this set of experiments the following three factors were varied: 1) control model, 2) network topology, and 3) communication link bandwidth. The values utilized in this set of experiments are presented in Table 6. Experiments employing all possible combinations of these factors were run.

6.1.2 Observations

Values for the average response time for a work request for all group 1 experiments are provided in Table 7. For each network topology a plot of average work request response time versus bandwidth for all models is provided in Figures 42 through 46. In order to aid in the analysis of this data, both absolute and relative differences of the response time values among the different control models have been computed and can be found in Table 8. Absolute and relative differences in response time values discovered at varying bandwidths with a single model of control are displayed in Table 9.

The comparison of response time results among the different control models indicates no significant variance for values obtained at bandwidths greater than 200 bytes/sec for all topologies. The unidirectional ring does not provide a significant variation until 200 bytes/sec. Experiments using the star and tree topologies provide significant variations only when the bandwidth is reduced to 50 and 100 bytes/sec, while both the fully connected and bidirectional ring topologies provided variations only at 50 bytes/sec.

Table 6. Variables for the Group 1 Experiments

Control Models

XFDPS.1
XFDPS.2
XFDPS.3
XFDPS.4

Network Topology

Unidirectional Ring
Bidirectional Ring
Star
Fully Connected
Tree

Communication Link Bandwidth

50 bytes/sec
100 bytes/sec
200 bytes/sec
600 bytes/sec
1,200 bytes/sec
50,000 bytes/sec
100,000 bytes/sec
500,000 bytes/sec
2,500,000 bytes/sec

The ordering of control models according to their average response times does not characterize a pattern. With a unidirectional ring topology and bandwidth of 100 bytes/sec, the ordering from longest response time on the left to shortest on the right is as follows:

XFDPS.1 > XFDPS.3 > XFDPS.2 > XFDPS.4

The bidirectional ring topology with a communication bandwidth of 50 bytes/sec provides similar results with the exception that the response times for XFDPS.2 and XFDPS.3 do not differ significantly.

XFDPS.1 > XFDPS.3 = XFDPS.2 > XFDPS.4

The star topology at both 50 and 100 bytes/sec provides an ordering in which the response time for XFDPS.2 is less than those for the other models, which show very little variation among themselves.

XFDPS.1 = XFDPS.3 = XFDPS.2 > XFDPS.4

Table 7. Average Work Request Response Time for Group 1

Unidirectional Ring

Bandwidth	XFDPS.1	XFDPS.2	XFDPS.3	XFDPS.4
50	182.3	210.3	250.4	111.4
100	169.7	103.1	141.3	63.0
200	92.8	53.7	82.5	48.5
600	47.9	41.3	45.2	45.6
1,200	45.0	47.1	43.6	48.3
50,000	48.2	44.9	39.1	45.2
100,000	41.6	47.4	43.5	48.1
500,000	35.7	49.4	45.6	46.3
2,500,000	42.2	45.4	45.2	44.4

Bidirectional Ring

Bandwidth	XFDPS.1	XFDPS.2	XFDPS.3	XFDPS.4
50	109.4	93.3	99.9	80.4
100	57.6	63.1	54.4	56.2
200	48.8	48.1	45.9	49.1
600	44.2	41.5	40.4	44.9
1,200	40.5	43.1	49.2	45.5
50,000	43.3	41.7	39.3	38.6
100,000	47.5	43.1	40.4	38.8
500,000	42.5	44.0	47.9	44.9
2,500,000	47.7	51.3	42.8	43.0

Star

Bandwidth	XFDPS.1	XFDPS.2	XFDPS.3	XFDPS.4
50	133.2	58.7	114.5	125.0
100	66.4	43.0	59.4	64.7
200	44.3	45.0	45.9	53.6
600	46.8	45.4	39.9	46.2
1,200	46.5	41.9	39.5	44.4
50,000	41.4	43.5	45.9	40.7
100,000	45.0	45.9	44.7	44.9
500,000	39.9	46.2	44.9	48.1
2,500,000	43.0	40.9	36.2	43.8

Note: all values are in seconds

Table 7. Average Work Request Response Time for Group 1
(continued)

Fully Connected

Bandwidth	XFDPS.1	XFDPS.2	XFDPS.3	XFDPS.4
50	47.7	47.6	47.2	68.3
100	43.8	51.4	42.8	51.3
200	46.7	47.0	44.5	47.3
600	42.6	42.9	47.2	47.4
1,200	43.2	46.3	45.1	43.3
50,000	44.0	39.7	39.9	44.2
100,000	44.4	38.2	36.3	45.4
500,000	42.8	46.1	43.1	43.5
2,500,000	41.3	49.2	43.6	41.2

Tree

Bandwidth	XFDPS.1	XFDPS.2	XFDPS.3	XFDPS.4
50	190.4	132.7	154.6	134.8
100	93.4	66.0	95.1	72.8
200	51.0	45.4	47.7	52.2
600	47.9	43.9	47.0	45.8
1,200	44.4	45.5	45.7	46.3
50,000	44.5	42.0	43.9	43.5
100,000	46.4	42.1	43.3	36.3
500,000	43.3	45.6	45.0	42.2
2,500,000	45.4	48.2	43.8	45.0

Note: all values are in seconds

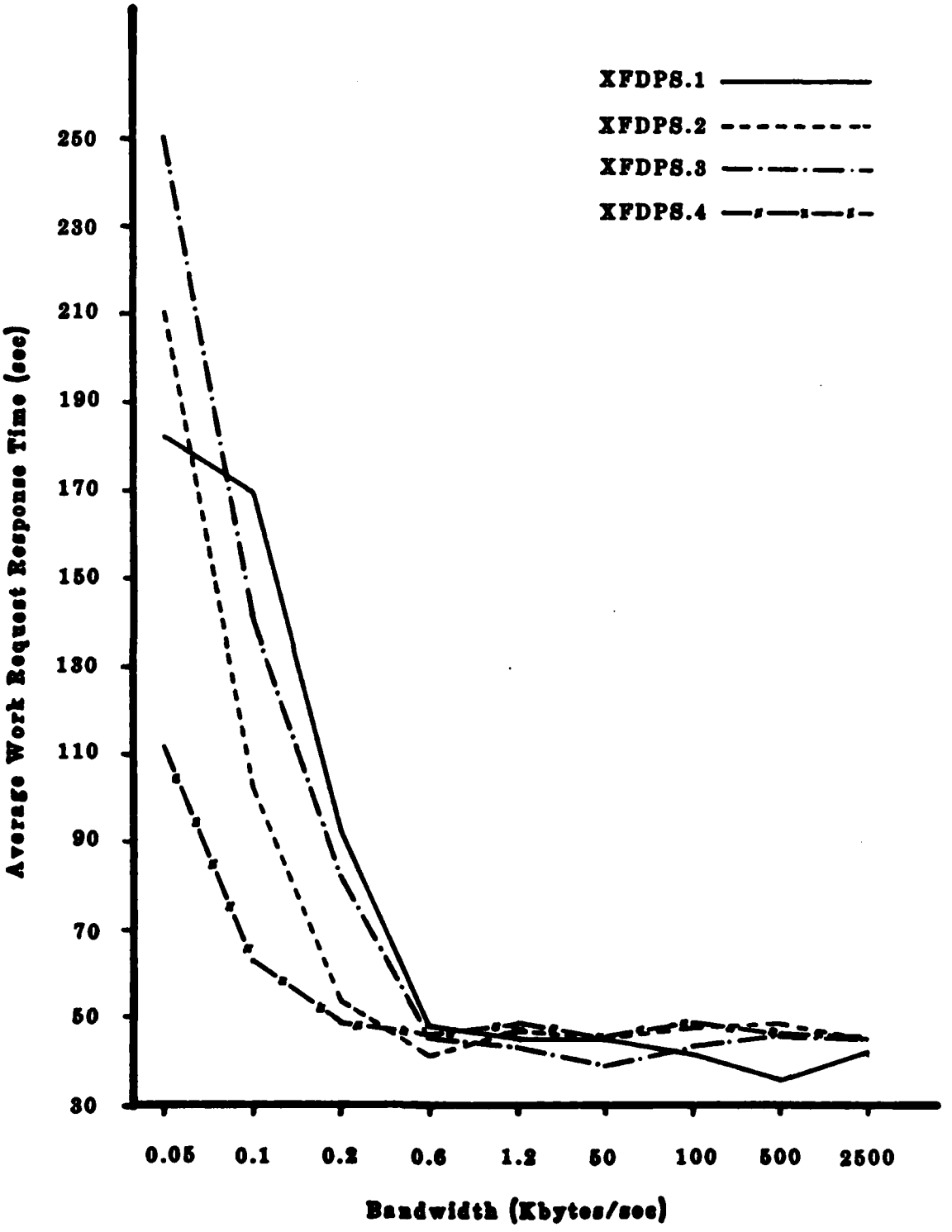


Figure 42. Average Work Request Response Time vs. Bandwidth for a Unidirectional Ring Network Topology for Group 1 Experiments

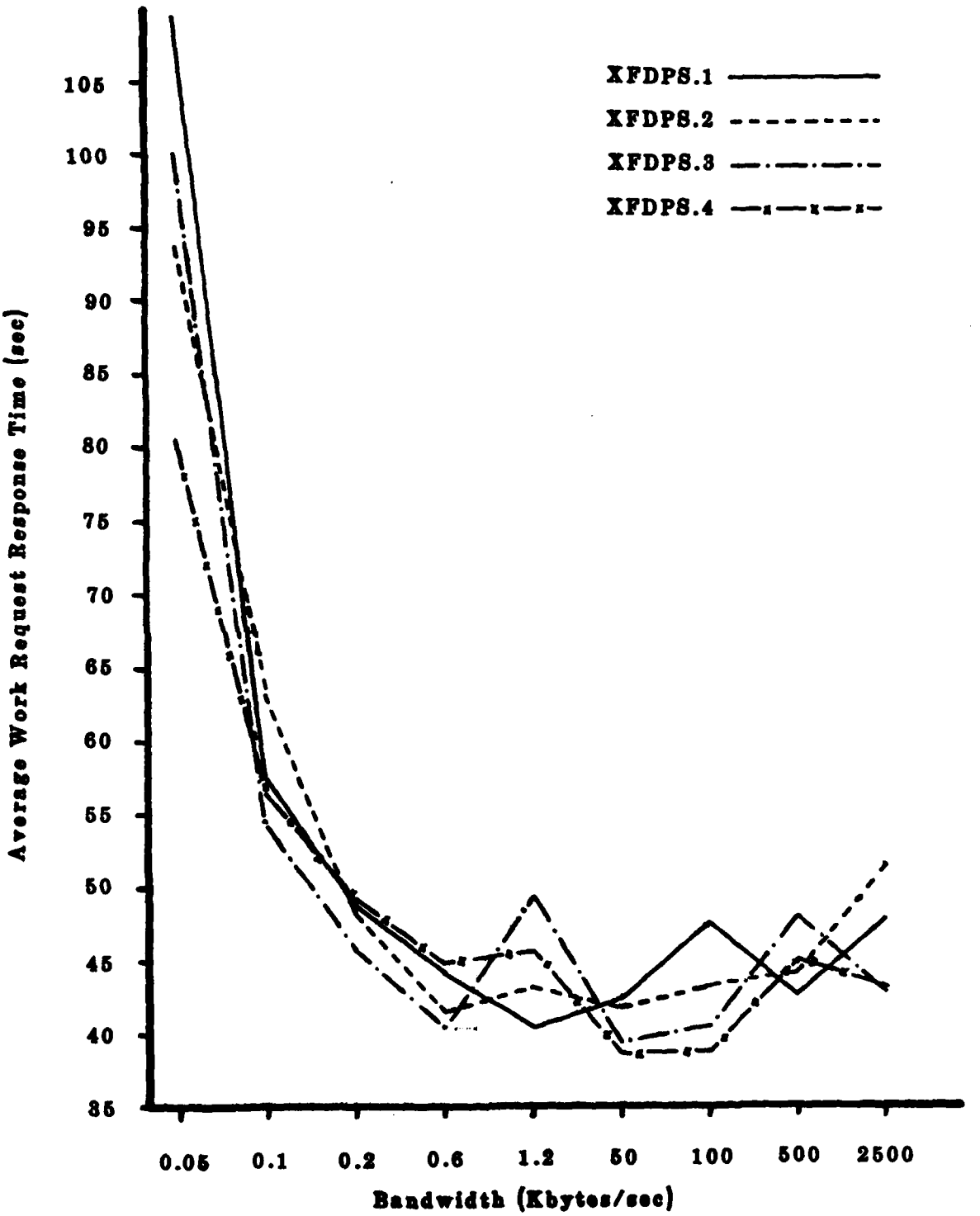


Figure 43. Average Work Request Response Time vs. Bandwidth
for a Bidirectional Ring Network Topology
for Group 1 Experiments

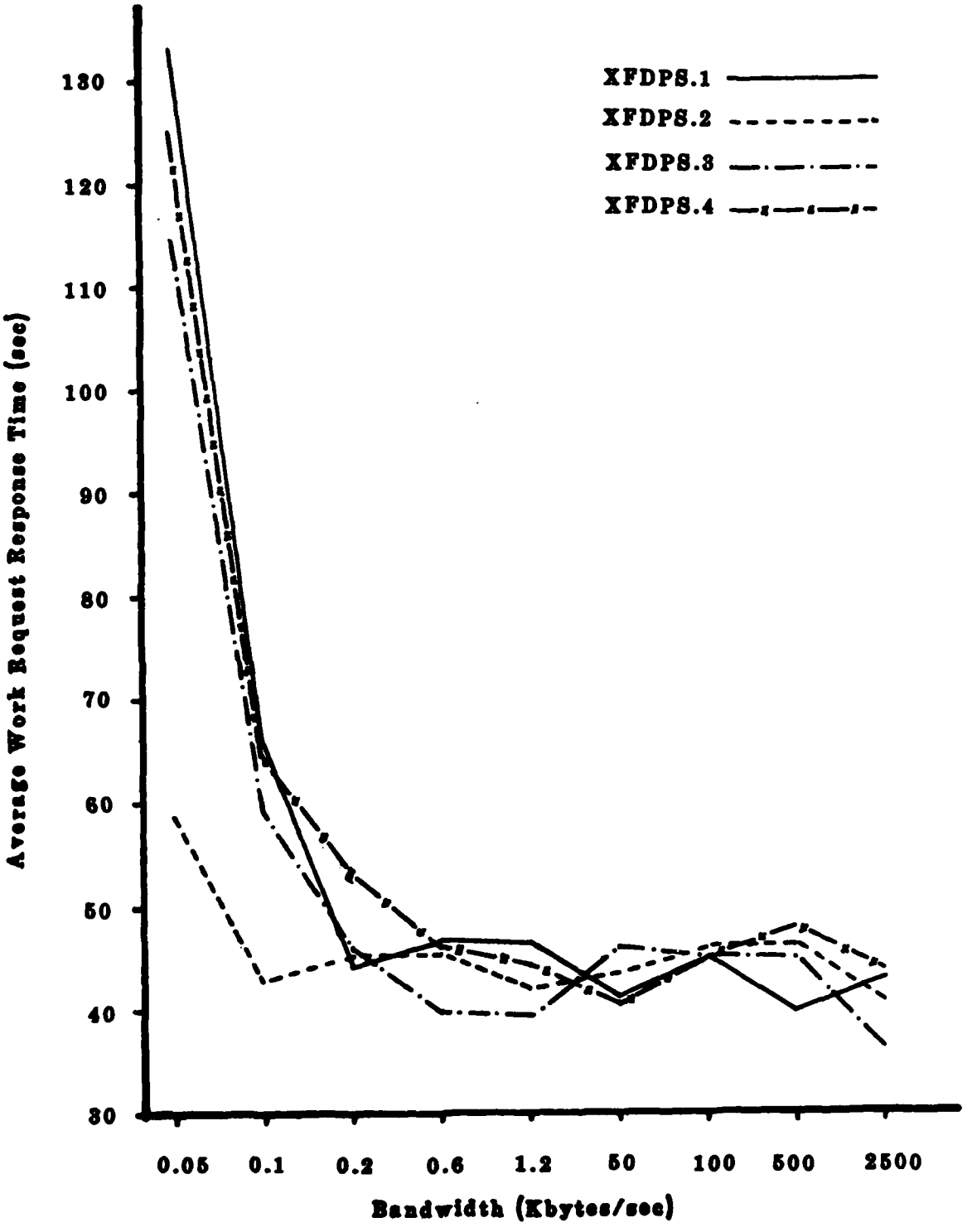


Figure 44. Average Work Request Response Time vs. Bandwidth
for a Star Network Topology
for Group 1 Experiments

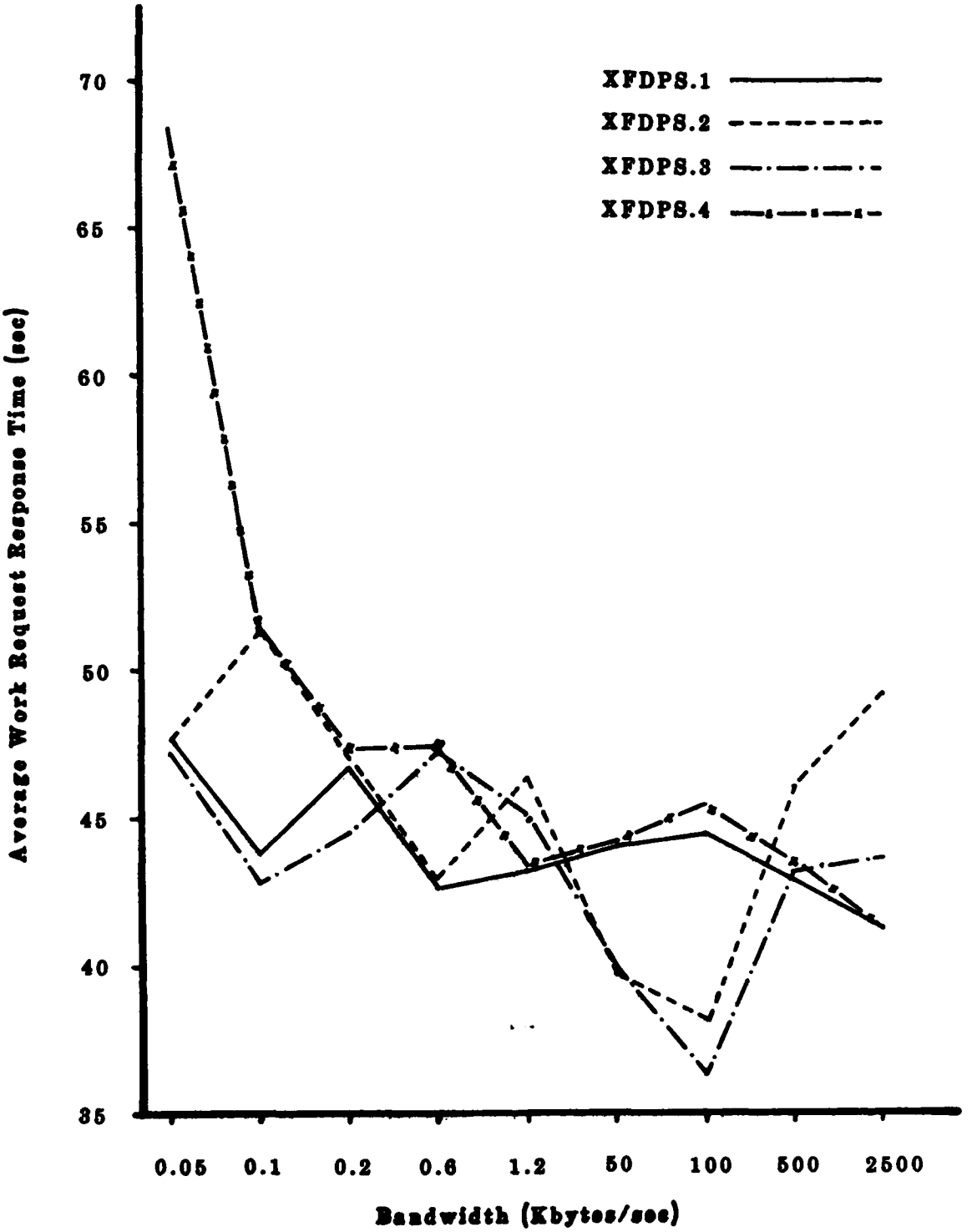


Figure 45. Average Work Request Response Time vs. Bandwidth for a Fully Connected Network Topology for Group 1 Experiments

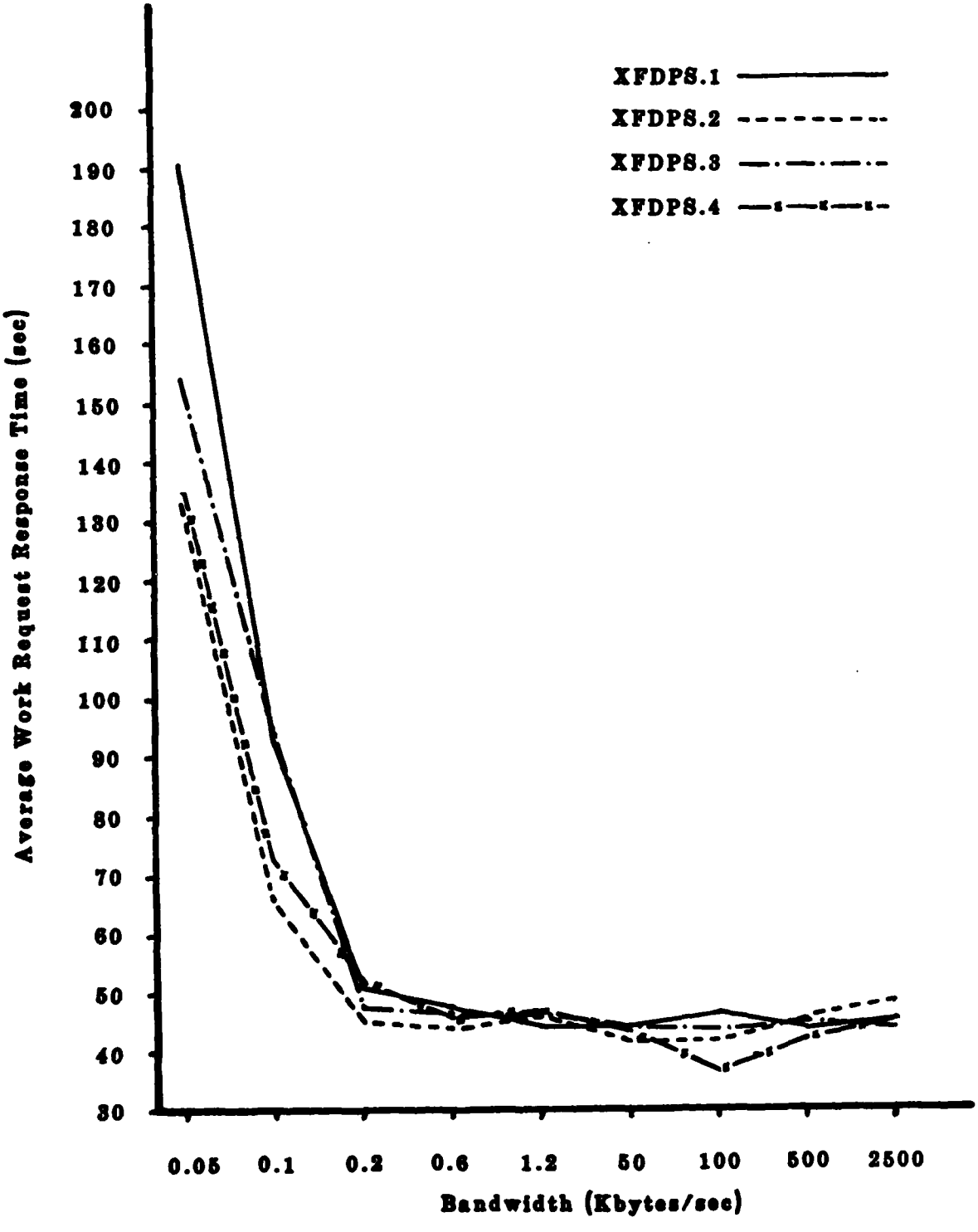


Figure 46. Average Work Request Response Time vs. Bandwidth
for a Tree Network Topology
for Group 1 Experiments

Table 8. Comparison of Response Times from Group 1 Experiments Using Different Control Models

Absolute Differences

Unidirectional Ring

Bandwidth	d12	d13	d14	d23	d24	d34
50	28.0	68.1	70.9	40.1	98.9	139.0
100	66.6	28.4	106.7	38.2	40.1	78.3
200	39.1	10.3	44.3	28.8	5.2	34.0
600	6.6	2.7	2.3	3.9	4.3	0.4
1,200	2.1	1.4	3.3	3.5	1.2	4.7
50,000	3.3	9.1	3.0	5.8	0.3	6.1
100,000	5.8	1.9	6.5	3.9	0.7	4.6
500,000	13.7	9.9	10.6	3.8	3.1	0.7
2,500,000	3.2	3.0	2.2	0.2	1.0	0.8

Bidirectional Ring

Bandwidth	d12	d13	d14	d23	d24	d34
50	16.1	9.5	29.0	6.6	12.9	19.5
100	5.5	3.2	1.4	8.7	6.9	1.8
200	0.7	2.9	0.3	2.2	1.0	3.2
600	2.7	3.8	0.7	1.1	3.4	4.5
1,200	2.6	8.7	5.0	6.1	2.4	3.7
50,000	1.6	4.0	4.7	2.4	3.1	0.7
100,000	4.4	7.1	8.7	2.7	4.3	1.6
500,000	1.5	5.4	2.4	3.9	0.9	3.0
2,500,000	3.6	4.9	4.7	8.5	8.3	0.2

Star

Bandwidth	d12	d13	d14	d23	d24	d34
50	74.5	18.7	8.2	55.8	66.3	10.5
100	23.4	7.0	1.7	16.4	21.7	5.3
200	0.7	1.6	9.3	0.9	8.6	7.7
600	1.4	6.9	0.6	5.5	0.8	6.3
1,200	4.6	7.0	2.1	2.4	2.5	4.9
50,000	2.1	4.5	0.7	2.4	2.8	5.2
100,000	0.9	0.3	0.1	1.2	1.0	0.2
500,000	6.3	5.0	8.2	1.3	1.9	3.2
2,500,000	2.1	6.8	0.8	4.7	2.9	7.6

Notation: dij = |RTi - RTj|, where RTi = Response time using XFDPS.i

(continued on next page)

Table 8. Comparison of Response Times from Group 1 Experiments Using Different Control Models (continued)

Absolute Differences

Fully Connected

Bandwidth	d12	d13	d14	d23	d24	d34
50	0.1	0.5	20.6	0.4	20.7	21.1
100	7.6	1.0	7.5	8.6	0.1	8.5
200	0.3	2.2	0.6	2.5	0.3	2.8
600	0.3	4.6	4.8	4.3	4.5	0.2
1,200	3.1	1.9	0.1	1.2	3.0	1.8
50,000	4.3	4.1	0.2	0.2	4.5	4.3
100,000	6.2	8.1	1.0	1.9	7.2	9.1
500,000	3.3	0.3	0.7	3.0	2.6	0.4
2,500,000	7.9	2.3	0.1	5.6	8.0	2.4

Tree

Bandwidth	d12	d13	d14	d23	d24	d34
50	57.7	35.8	55.6	21.9	2.1	19.8
100	27.4	1.7	20.6	29.1	6.8	22.3
200	5.6	3.3	1.2	2.3	6.8	4.5
600	4.0	0.9	2.1	3.1	1.9	1.2
1,200	1.1	1.3	1.9	0.2	0.8	0.6
50,000	2.5	0.6	1.0	1.9	1.5	0.4
100,000	4.3	3.1	10.1	1.2	5.8	7.0
500,000	2.3	1.7	1.1	0.6	3.4	2.8
2,500,000	2.8	1.6	0.4	4.4	3.2	1.2

Notation: dij = |RTi - RTj|, where RTi = Response time using XFDPS.i

(continued on next page)

Table 8. Comparison of Response Times from Group 1 Experiments Using Different Control Models (continued)

Relative Differences

Unidirectional Ring

Bandwidth	d12	d13	d14	d23	d24	d34
50	0.13	0.27	0.39	0.16	0.47	0.56
100	0.39	0.17	0.63	0.27	0.39	0.55
200	0.42	0.11	0.48	0.35	0.10	0.41
600	0.14	0.06	0.05	0.09	0.09	0.00
1,200	0.04	0.03	0.07	0.07	0.02	0.10
50,000	0.07	0.19	0.06	0.13	0.00	0.13
100,000	0.12	0.04	0.14	0.08	0.01	0.10
500,000	0.28	0.22	0.23	0.08	0.06	0.02
2,500,000	0.07	0.07	0.05	0.00	0.02	0.02

Bidirectional Ring

Bandwidth	d12	d13	d14	d23	d24	d34
50	0.15	0.09	0.27	0.07	0.14	0.20
100	0.09	0.06	0.02	0.14	0.11	0.03
200	0.01	0.06	0.00	0.05	0.02	0.07
600	0.06	0.09	0.02	0.03	0.08	0.10
1,200	0.06	0.18	0.11	0.12	0.05	0.08
50,000	0.04	0.09	0.11	0.06	0.07	0.02
100,000	0.09	0.15	0.18	0.06	0.10	0.04
500,000	0.03	0.11	0.05	0.08	0.02	0.06
2,500,000	0.07	0.10	0.10	0.17	0.16	0.00

Star

Bandwidth	d12	d13	d14	d23	d24	d34
50	0.56	0.14	0.06	0.49	0.53	0.08
100	0.35	0.11	0.03	0.28	0.34	0.08
200	0.02	0.03	0.17	0.02	0.16	0.14
600	0.03	0.15	0:01	0.12	0.02	0.14
1,200	0.10	0.15	0.05	0.06	0.06	0.11
50,000	0.05	0.10	0.02	0.05	0.06	0.11
100,000	0.02	0.00	0.00	0.03	0.02	0.00
500,000	0.14	0.11	0.17	0.03	0.04	0.07
2,500,000	0.05	0.16	0.02	0.11	0.07	0.17

Notation: $dij = |RTi - RTj| / Max(RTi, RTj)$

(continued on next page)

Table 8. Comparison of Response Times from Group 1 Experiments Using Different Control Models (continued)

Relative Differences

Fully Connected

Bandwidth	d12	d13	d14	d23	d24	d34
50	0.00	0.01	0.30	0.00	0.30	0.31
100	0.15	0.02	0.15	0.17	0.00	0.17
200	0.00	0.05	0.01	0.05	0.00	0.06
600	0.00	0.10	0.10	0.09	0.09	0.00
1,200	0.07	0.04	0.00	0.03	0.06	0.04
50,000	0.10	0.09	0.00	0.00	0.10	0.10
100,000	0.14	0.18	0.02	0.05	0.16	0.20
500,000	0.07	0.00	0.02	0.07	0.06	0.00
2,500,000	0.16	0.05	0.00	0.11	0.16	0.06

Tree

Bandwidth	d12	d13	d14	d23	d24	d34
50	0.30	0.19	0.29	0.14	0.02	0.13
100	0.29	0.02	0.22	0.31	0.09	0.23
200	0.11	0.06	0.02	0.05	0.13	0.09
600	0.08	0.02	0.04	0.07	0.04	0.03
1,200	0.02	0.03	0.04	0.00	0.02	0.01
50,000	0.06	0.01	0.02	0.04	0.03	0.00
100,000	0.09	0.07	0.22	0.03	0.14	0.16
500,000	0.05	0.04	0.03	0.01	0.07	0.06
2,500,000	0.06	0.04	0.00	0.09	0.07	0.03

Notation: $dij = |RTi - RTj| / Max (RTi, RTj)$

Table 9. Comparison of Response Times from Group 1 Experiments Using Different Bandwidths but the Same Control Model

Absolute Differences

Unidirectional Ring

Bandwidths a	Bandwidths b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
50	100	12.6	107.2	109.1	48.4
100	200	76.9	49.4	58.8	14.5
200	600	44.9	12.4	37.3	2.9
600	1,200	2.9	5.8	1.6	2.7
1,200	50,000	3.2	2.2	4.5	3.1
50,000	100,000	6.6	2.5	4.4	2.9
100,000	500,000	5.9	2.0	2.1	1.8
500,000	2,500,000	6.5	4.0	0.4	1.9

Bidirectional Ring

Bandwidths a	Bandwidths b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
50	100	51.8	30.2	45.5	24.2
100	200	8.8	15.0	8.5	7.1
200	600	4.6	6.6	5.5	4.2
600	1,200	3.7	1.6	8.8	0.6
1,200	50,000	2.8	1.4	9.9	6.9
50,000	100,000	4.2	1.4	1.1	0.2
100,000	500,000	5.0	0.9	7.5	6.1
500,000	2,500,000	5.2	7.3	5.1	1.9

Star

Bandwidths a	Bandwidths b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
50	100	66.8	15.7	55.1	60.3
100	200	22.1	2.0	13.5	11.1
200	600	2.5	0.4	6.0	7.4
600	1,200	0.3	3.5	0.4	1.8
1,200	50,000	5.1	1.6	6.4	3.7
50,000	100,000	3.6	2.4	1.2	4.2
100,000	500,000	5.1	0.3	0.2	3.2
500,000	2,500,000	3.1	5.3	8.7	4.3

(continued on next page)

Table 9. Comparison of Response Times from Group 1 Experiments Using Different Control Models but the Same Bandwidth (continued)

Fully Connected

Bandwidths a	b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
50	100	3.9	3.8	4.4	17.0
100	200	2.9	4.4	1.7	4.0
200	600	4.1	4.1	2.7	0.1
600	1,200	0.6	3.4	2.1	4.1
1,200	50,000	0.8	6.6	5.2	0.9
50,000	100,000	0.4	1.5	3.6	1.2
100,000	500,000	1.6	7.9	6.8	1.9
500,000	2,500,000	1.5	3.1	0.5	2.3

Tree

Bandwidths a	b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
50	100	97.0	66.7	59.5	62.0
100	200	42.4	20.6	47.4	20.6
200	600	3.1	1.5	0.7	6.4
600	1,200	3.5	1.6	1.3	0.5
1,200	50,000	0.1	3.5	1.8	2.8
50,000	100,000	1.9	0.1	0.6	7.2
100,000	500,000	3.1	3.5	1.7	5.9
500,000	2,500,000	2.1	2.6	1.2	2.8

Relative Differences

Unidirectional Ring

Bandwidths a	b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
50	100	0.07	0.51	0.44	0.43
100	200	0.45	0.48	0.42	0.23
200	600	0.48	0.23	0.45	0.06
600	1,200	0.06	0.12	0.04	0.06
1,200	50,000	0.07	0.05	0.10	0.06
50,000	100,000	0.14	0.05	0.10	0.06
100,000	500,000	0.14	0.04	0.05	0.04
500,000	2,500,000	0.15	0.08	0.00	0.04

(continued on next page)

Table 9. Comparison of Response Times from Group 1 Experiments Using Different Control Models but the Same Bandwidth (continued)

Bidirectional Ring

Bandwidths a	Bandwidths b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
50	100	0.47	0.32	0.46	0.30
100	200	0.15	0.24	0.16	0.13
200	600	0.09	0.14	0.12	0.09
600	1,200	0.08	0.04	0.18	0.01
1,200	50,000	0.06	0.03	0.20	0.15
50,000	100,000	0.09	0.03	0.03	0.00
100,000	500,000	0.11	0.02	0.16	0.14
500,000	2,500,000	0.11	0.14	0.11	0.04

Star

Bandwidths a	Bandwidths b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
50	100	0.50	0.27	0.48	0.48
100	200	0.33	0.04	0.23	0.17
200	600	0.05	0.00	0.13	0.14
600	1,200	0.00	0.08	0.01	0.04
1,200	50,000	0.11	0.04	0.14	0.08
50,000	100,000	0.08	0.05	0.03	0.09
100,000	500,000	0.11	0.00	0.00	0.07
500,000	2,500,000	0.07	0.11	0.19	0.09

Fully Connected

Bandwidths a	Bandwidths b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
50	100	0.08	0.07	0.09	0.25
100	200	0.06	0.09	0.04	0.08
200	600	0.09	0.09	0.06	0.00
600	1,200	0.01	0.07	0.04	0.09
1,200	50,000	0.02	0.14	0.12	0.02
50,000	100,000	0.00	0.04	0.09	0.03
100,000	500,000	0.04	0.17	0.16	0.04
500,000	2,500,000	0.04	0.06	0.01	0.05

(continued on next page)

Table 9. Comparison of Response Times from Group 1 Experiments Using Different Control Models but the Same Bandwidth (continued)

Bandwidths a	Bandwidths b	Tree XFDPS.1 dab	Tree XFDPS.2 dab	Tree XFDPS.3 dab	Tree XFDPS.4 dab
50	100	0.51	0.50	0.38	0.46
100	200	0.45	0.31	0.50	0.28
200	600	0.06	0.03	0.01	0.12
600	1,200	0.07	0.04	0.03	0.01
1,200	50,000	0.00	0.08	0.04	0.06
50,000	100,000	0.04	0.00	0.01	0.17
100,000	500,000	0.07	0.08	0.04	0.14
500,000	2,500,000	0.05	0.05	0.03	0.06

It is XFDPS.4 that provides a larger average response time than the other models when a fully connected topology with a bandwidth of 50 bytes/sec is utilized.

XFDPS.4 > XFDPS.1 = XFDPS.2 = XFDPS.3

The tree topology at 50 and 100 bytes/sec provides results indicating superior performance by XFDPS.2 and XFDPS.4 over that of XFDPS.1 and XFDPS.3.

XFDPS.1 > XFDPS.3 > XFDPS.2 = XFDPS.4

A comparison of the results of each model at different bandwidths indicates very little variation until a relatively small bandwidth is reached. With the unidirectional ring no variation is observed until the bandwidth is changed from 600 to 200 bytes/sec. The point of change for the bidirectional ring and star topologies does not occur until the bandwidth is reduced from 100 to 50 bytes/sec. The tree topology shows a change when the bandwidth is changed from 200 to 100 bytes/sec. Only XFDPS.4 shows a change with a fully connected topology. This change occurs when the bandwidth is varied from 100 to 50 bytes/sec.

This group of experiments demonstrates very little variation in average response times. Only when the communication bandwidth is made very small is any appreciable variation observed. A comparison of results among the models

indicates no consistent pattern if the control models are ordered by the average response times obtained when each model is utilized.

6.2 Work Requests Utilizing Only Remote File Access

6.2.1 The Environment

The second group of experiments investigated the effect of the presence of user message traffic in addition to control message traffic on the communication system. The work requests in this set of experiments were similar to those used in the first group with the exception that the object and data files of an object-data file pair were located on different nodes. In all cases if the object file was located on node i, the data file was located on node j where

$$j = \begin{cases} i + 1, & i < 5 \\ 1, & i = 5 \end{cases}$$

As in the first group, the object-data file pairs were spread evenly across all nodes of the network.

The same three factors (control model, network topology, and communication link bandwidth) utilized in the first group of experiments were used in the second set. The values used in this set of experiments are presented in Table 10. Experiments utilizing all combinations of these factors were run.

6.2.2 Observations

Table 11 contains the average work request response times for the second group of experiments. As with the group 1 data, plots of average work request response time versus bandwidth for all control models are presented for each network topology in Figures 47 through 51. Table 12 contains a comparison of the average response time values obtained with different control models with the same network topology and communication bandwidth. A comparison of the average response time values obtained with the same control model using the same network topology but different communication bandwidths is provided in Table 13.

The results obtained with different control models does not provide any pattern in which the control models can be ordered according to average response times obtained with the models. Results from experiments using a

Table 10. Variables for the Group 2 Experiments

Control Models

XFDPS.1
XFDPS.2
XFDPS.3
XFDPS.4

Network Topology

Unidirectional Ring
Bidirectional Ring
Star
Fully Connected
Tree

Communication Link Bandwidth

50,000 bytes/sec
100,000 bytes/sec
500,000 bytes/sec

unidirectional ring network topology indicate similar performance characteristics for all models.

XFDPS.1 = XFDPS.2 = XFDPS.3 = XFDPS.4

Experiments utilizing the bidirectional ring indicate a different ordering at each bandwidth. At 50,000 bytes/sec the average response time for XFDPS.4 is longer than those for the other models which have similar values.

XFDPS.4 > XFDPS.1 = XFDPS.2 = XFDPS.3

No significant difference is found among the models at 100,000 bytes/sec. At 500,000 bytes/sec the average response times are ordered as follows:

XFDPS.1 = XFDPS.3 > XFDPS.2 > XFDPS.4

The star topology experiences changes only with a bandwidth of 500,000 bytes/sec. In this case the average response time for XFDPS.4 is greater than that obtained while utilizing the other models.

XFDPS.4 > XFDPS.1 = XFDPS.2 = XFDPS.3

At 50,000 and 500,000 bytes/sec, experiments with a fully connected network topology provide the same ordering in which the average response time for XFDPS.4 is larger than that obtained with any of the other models.

Table 11. Average Work Request Response Time for Group 2

Unidirectional Ring

Bandwidth	XFDPS.1	XFDPS.2	XFDPS.3	XFDPS.4
50,000	450.4	470.9	461.3	460.0
100,000	230.2	216.2	220.8	229.1
500,000	55.1	56.2	56.6	57.4

Bidirectional Ring

Bandwidth	XFDPS.1	XFDPS.2	XFDPS.3	XFDPS.4
50,000	78.3	70.0	77.0	61.7
100,000	49.6	47.2	52.4	55.0
500,000	54.2	49.6	48.9	61.5

Star

Bandwidth	XFDPS.1	XFDPS.2	XFDPS.3	XFDPS.4
50,000	122.4	124.5	120.0	121.1
100,000	59.1	58.8	57.9	64.1
500,000	54.4	52.9	50.4	60.0

Fully Connected

Bandwidth	XFDPS.1	XFDPS.2	XFDPS.3	XFDPS.4
50,000	71.1	66.8	69.2	84.1
100,000	54.3	48.9	47.7	57.0
500,000	48.9	49.2	50.0	61.5

Tree

Bandwidth	XFDPS.1	XFDPS.2	XFDPS.3	XFDPS.4
50,000	239.0	238.2	186.7	214.5
100,000	107.0	112.2	115.0	116.2
500,000	55.6	61.5	55.0	64.4

Note: all values are in seconds

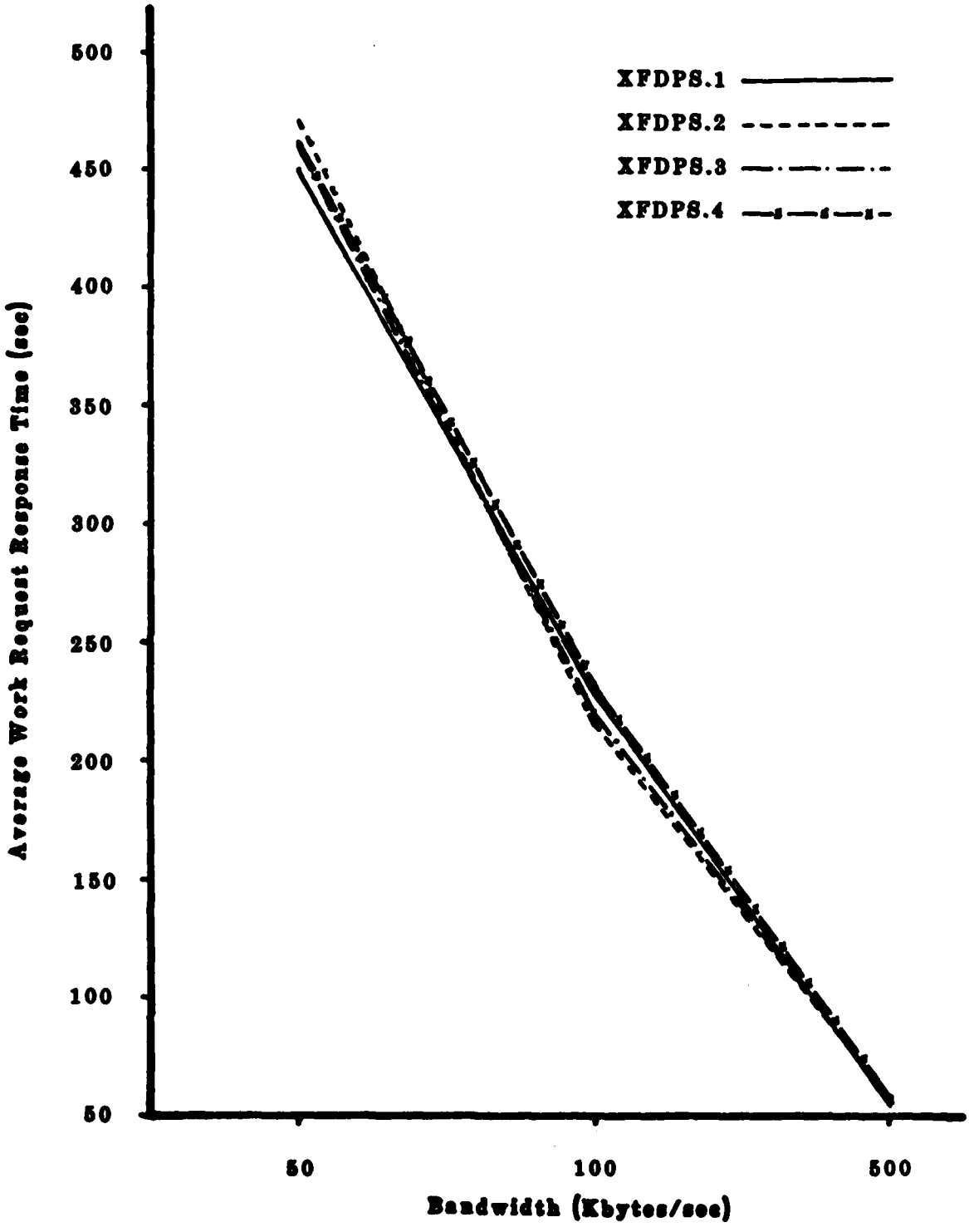


Figure 47. Average Work Request Response Time vs. Bandwidth for a Unidirectional Ring Network Topology for Group 2 Experiments

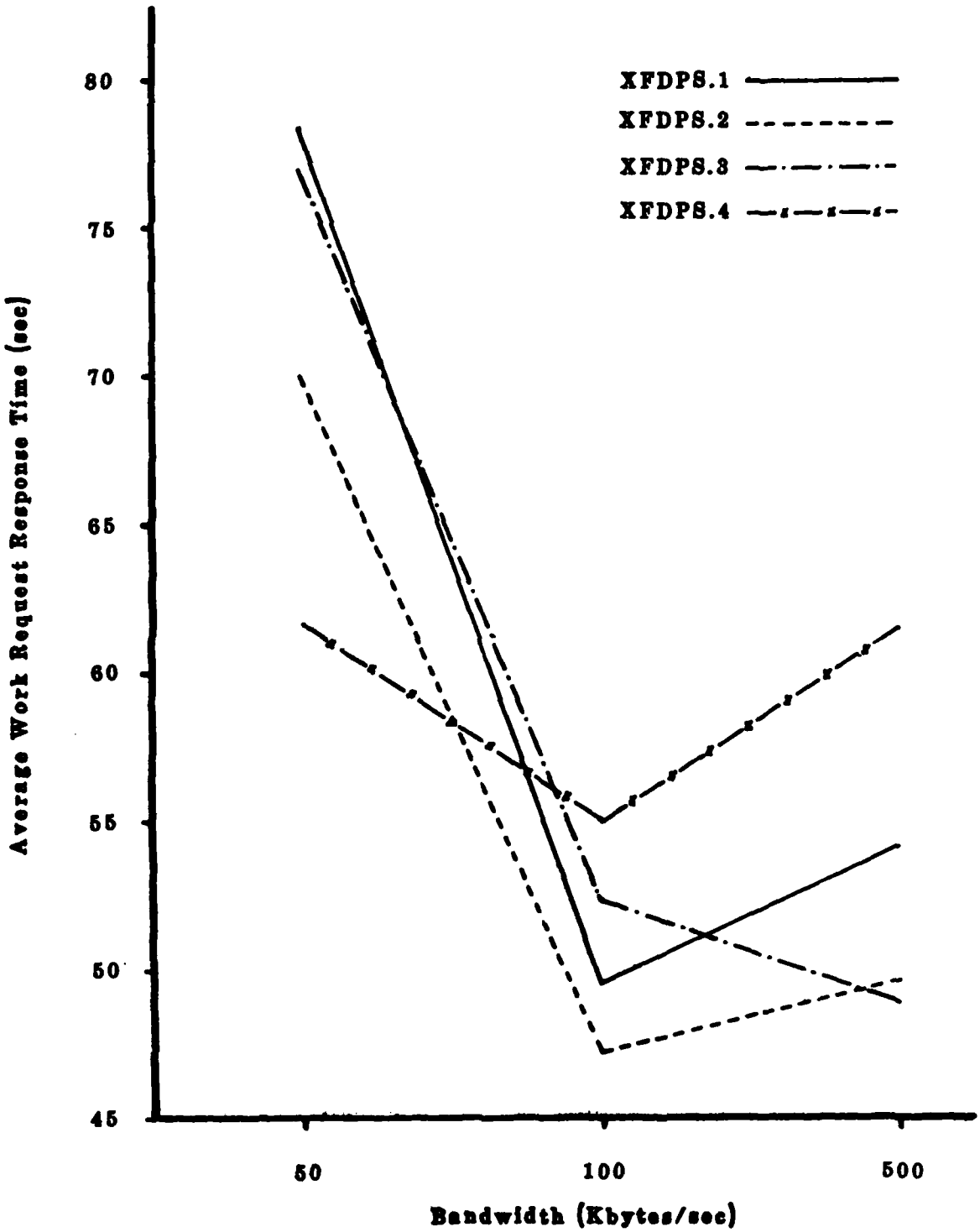


Figure 48. Average Work Request Response Time vs. Bandwidth for a Bidirectional Ring Network Topology for Group 2 Experiments

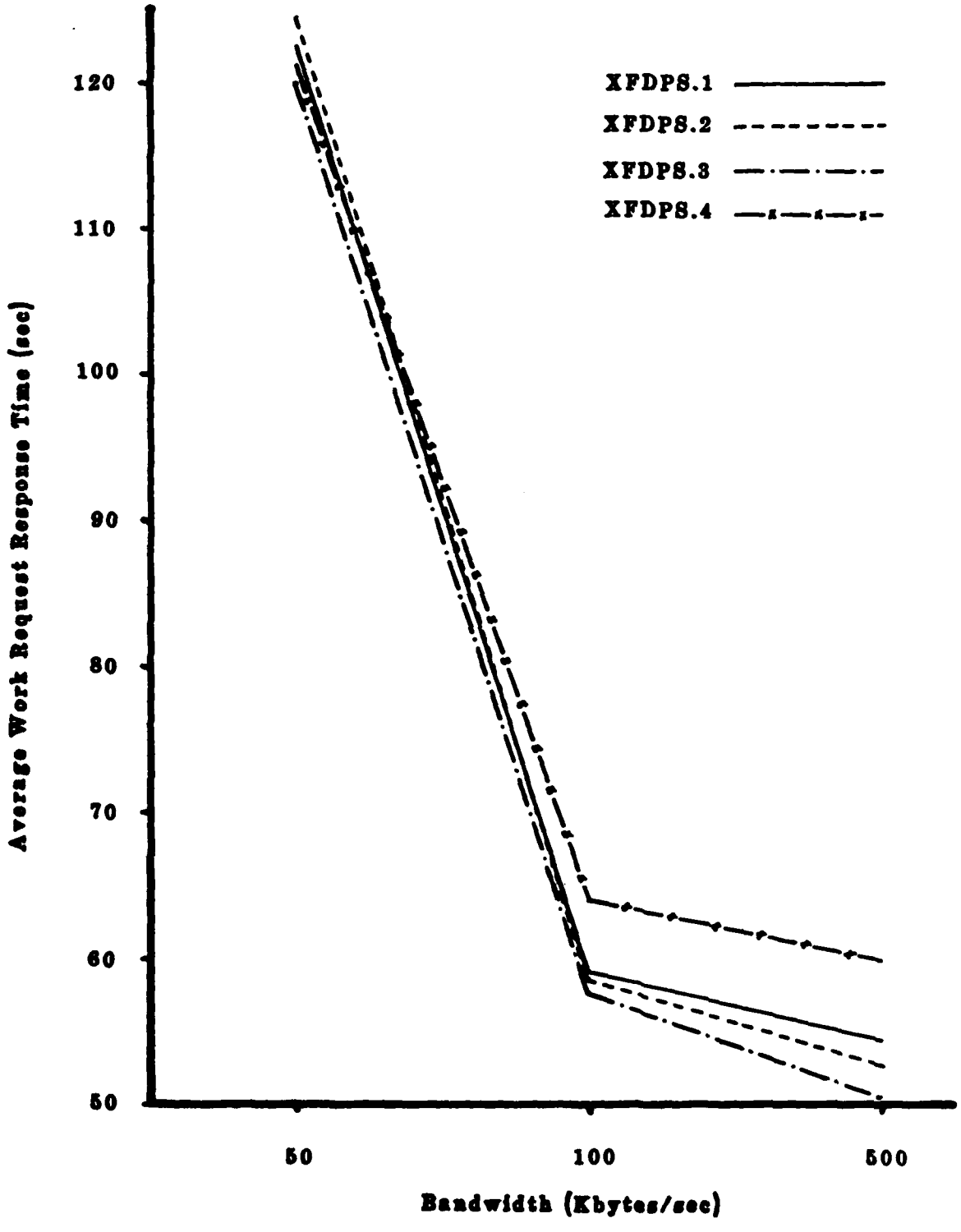


Figure 49. Average Work Request Response Time vs. Bandwidth
for a Star Network Topology
for Group 2 Experiments

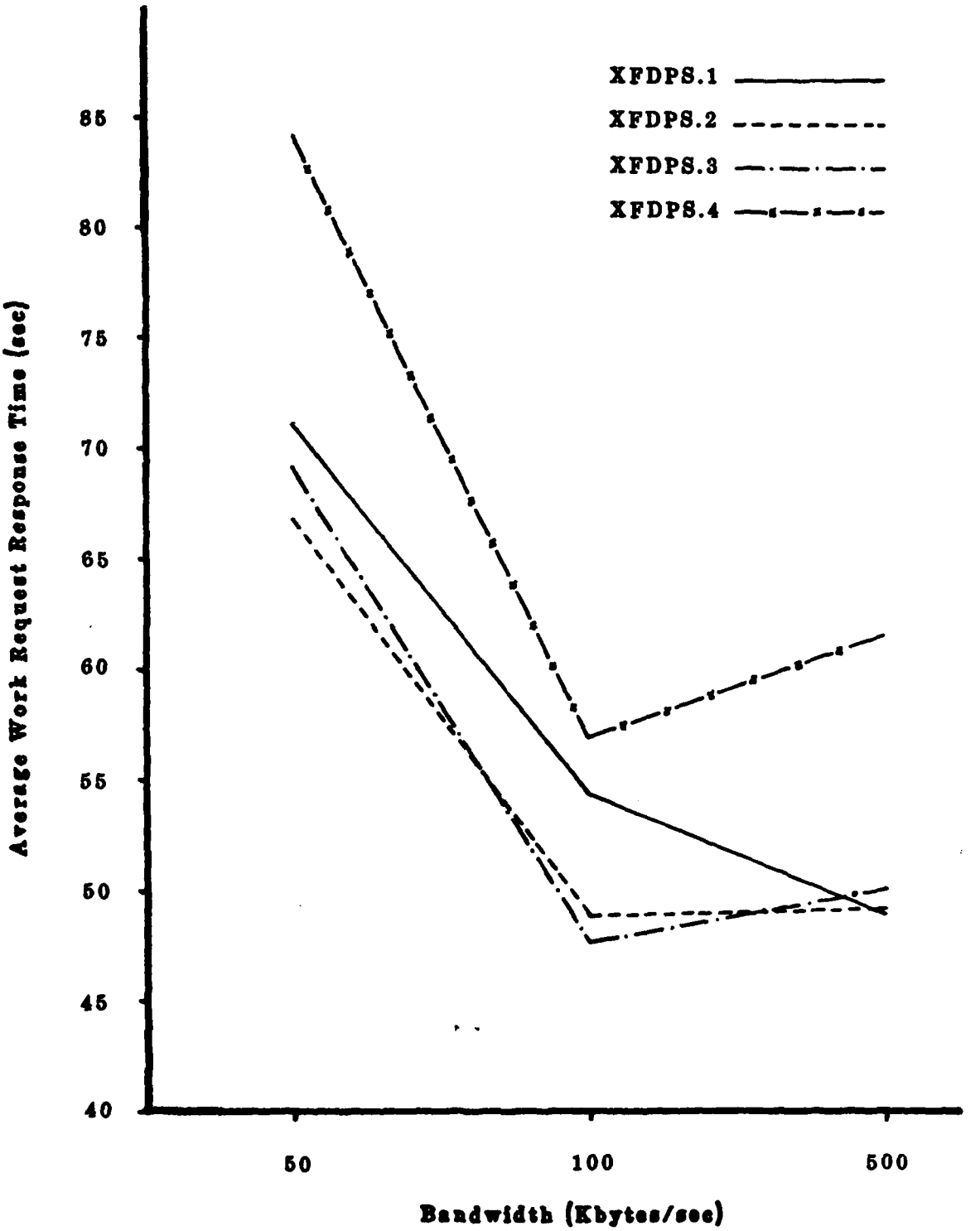


Figure 50. Average Work Request Response Time vs. Bandwidth
for a Fully Connected Network Topology
for Group 2 Experiments

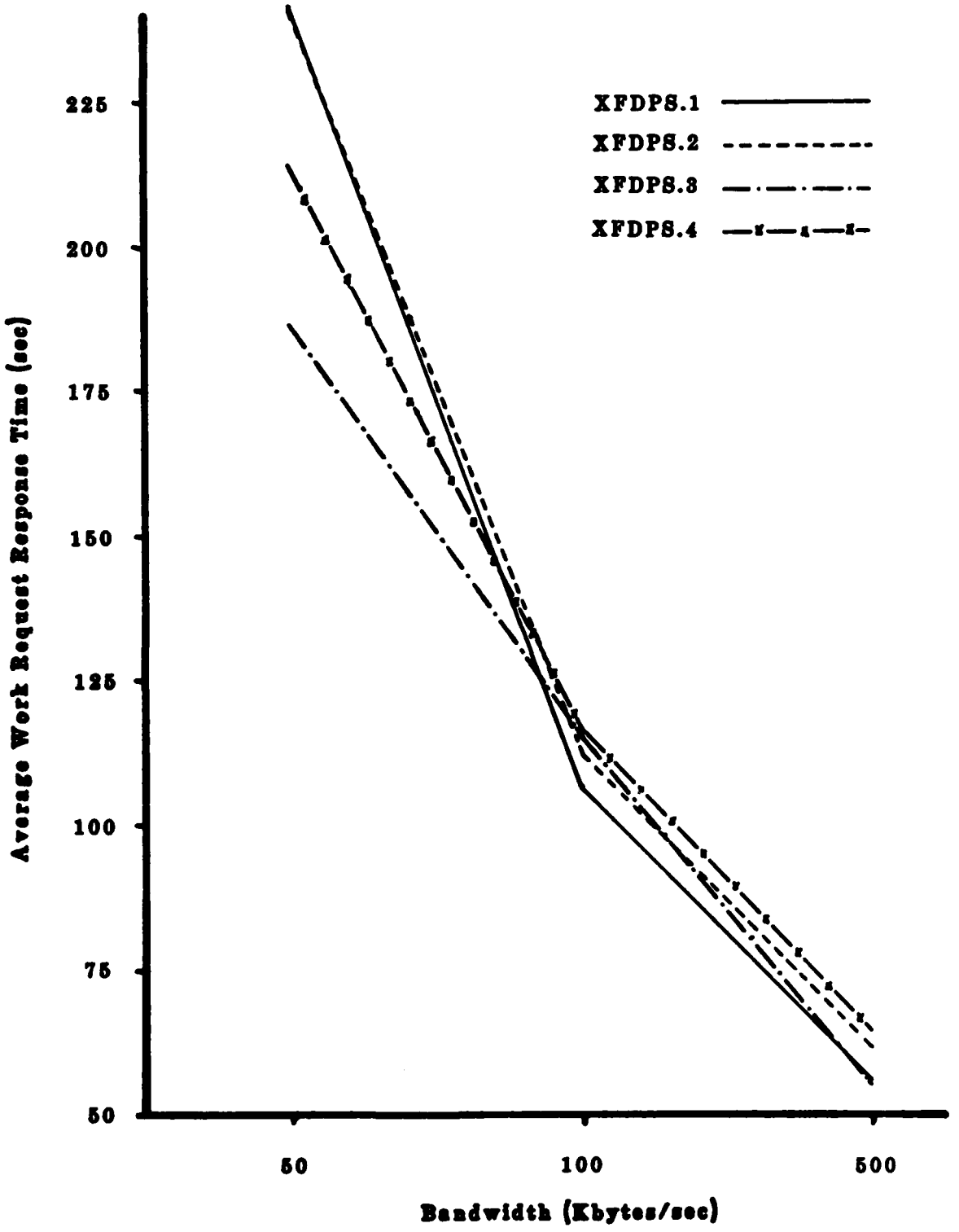


Figure 51. Average Work Request Response Time vs. Bandwidth
for a Tree Network Topology
for Group 2 Experiments

Table 12. Comparison of Response Times from Group 2 Experiments Using Different Control Models

Absolute Differences

Unidirectional Ring

Bandwidth	d12	d13	d14	d23	d24	d34
50,000	20.5	10.9	9.6	9.6	10.9	1.3
100,000	14.0	9.4	1.1	4.6	12.9	8.3
500,000	1.1	1.5	2.3	0.4	1.2	0.8

Bidirectional Ring

Bandwidth	d12	d13	d14	d23	d24	d34
50,000	8.3	1.3	16.6	7.0	8.3	15.3
100,000	2.4	2.8	5.4	5.2	7.8	2.6
500,000	4.6	5.3	7.3	0.7	11.9	12.6

Star

Bandwidth	d12	d13	d14	d23	d24	d34
50,000	2.1	2.4	1.3	4.5	3.4	1.1
100,000	0.3	1.2	5.0	0.9	5.3	6.2
500,000	1.5	4.0	5.6	2.5	7.1	9.6

Fully Connected

Bandwidth	d12	d13	d14	d23	d24	d34
50,000	4.3	1.9	13.0	2.4	17.3	14.9
100,000	5.4	6.6	2.7	1.2	8.1	9.3
500,000	0.3	1.1	12.6	0.8	12.3	11.5

Tree

Bandwidth	d12	d13	d14	d23	d24	d34
50,000	0.8	52.3	24.5	51.5	23.7	27.8
100,000	5.2	8.0	9.2	2.8	4.0	1.2
500,000	5.9	0.6	8.8	6.5	2.9	9.4

Notation: dij = |RTi - RTj|, where RTi = Response time using XFDPS.i

(continued on next page)

Table 12. Comparison of Response Times from Group 2 Experiments Using Different Control Models (continued)

Relative Differences

Unidirectional Ring

Bandwidth	d12	d13	d14	d23	d24	d34
50,000	0.04	0.02	0.02	0.02	0.02	0.00
100,000	0.06	0.04	0.00	0.02	0.06	0.04
500,000	0.02	0.03	0.04	0.00	0.02	0.01

Bidirectional Ring

Bandwidth	d12	d13	d14	d23	d24	d34
50,000	0.11	0.02	0.21	0.09	0.12	0.20
100,000	0.05	0.05	0.10	0.10	0.14	0.05
500,000	0.08	0.10	0.12	0.01	0.19	0.20

Star

Bandwidth	d12	d13	d14	d23	d24	d34
50,000	0.02	0.02	0.01	0.04	0.03	0.00
100,000	0.00	0.02	0.08	0.02	0.08	0.10
500,000	0.03	0.07	0.09	0.05	0.12	0.16

Fully Connected

Bandwidth	d12	d13	d14	d23	d24	d34
50,000	0.06	0.03	0.15	0.03	0.21	0.18
100,000	0.10	0.12	0.05	0.02	0.14	0.16
500,000	0.00	0.02	0.20	0.02	0.20	0.19

Tree

Bandwidth	d12	d13	d14	d23	d24	d34
50,000	0.00	0.22	0.10	0.22	0.10	0.13
100,000	0.05	0.07	0.08	0.02	0.03	0.01
500,000	0.10	0.01	0.14	0.11	0.05	0.15

Notation: $dij = |RTi - RTj| / Max(RTi, RTj)$

Table 13. Comparison of Response Times from Group 2 Experiments Using Different Bandwidths but the Same Control Model

Absolute Differences

Unidirectional Ring

Bandwidths a	b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
50,000	100,000	220.2	254.7	240.5	230.9
100,000	500,000	175.1	160.0	164.2	171.7

Bidirectional Ring

Bandwidths a	b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
50,000	100,000	28.7	22.8	24.6	6.7
100,000	500,000	4.6	2.4	3.5	6.5

Star

Bandwidths a	b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
50,000	100,000	63.3	65.7	62.1	57.0
100,000	500,000	4.7	5.9	7.5	4.1

Fully Connected

Bandwidths a	b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
50,000	100,000	16.8	17.9	21.5	27.1
100,000	500,000	5.4	0.3	2.3	4.5

Tree

Bandwidths a	b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
50,000	100,000	132.0	126.0	71.7	98.3
100,000	500,000	51.4	50.7	60.0	51.8

(continued on next page)

Table 13. Comparison of Response Times from Group 1 Experiments Using Different Control Models but the Same Bandwidth (continued)

Relative Differences

Unidirectional Ring

Bandwidths a	b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
50,000	100,000	0.49	0.54	0.52	0.50
100,000	500,000	0.76	0.74	0.74	0.75

Bidirectional Ring

Bandwidths a	b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
50,000	100,000	0.37	0.33	0.32	0.11
100,000	500,000	0.08	0.05	0.07	0.11

Star

Bandwidths a	b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
50,000	100,000	0.52	0.53	0.52	0.47
100,000	500,000	0.08	0.10	0.13	0.06

Fully Connected

Bandwidths a	b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
50,000	100,000	0.24	0.27	0.31	0.32
100,000	500,000	0.10	0.00	0.05	0.07

Tree

Bandwidths a	b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
50,000	100,000	0.55	0.53	0.38	0.46
100,000	500,000	0.48	0.45	0.52	0.45

XFDPS.4 > XFDPS.1 = XFDPS.2 = XFDPS.3

At 100,000 bytes/sec XFDPS.1 and XFDPS.4 provide similar values which are larger than those provided by XFDPS.2 and XFDPS.3.

XFDPS.1 = XFDPS.4 > XFDPS.2 = XFDPS.3

The tree network topology displays differences only with a bandwidth of 50,000 bytes/sec. The ordering in this case is as follows:

XFDPS.1 = XFDPS.2 > XFDPS.3 > XFDPS.4

A comparison of average response time values obtained with the same control model and network topology but varying the bandwidth indicates changes at all bandwidths for all models when both the unidirectional ring and tree network topologies are used. Experiments utilizing XFDPS.4 and the bidirectional ring network topology demonstrate no significant variance among the results obtained with the three bandwidths (50,000, 100,000, and 500,000 bytes/sec) used in these experiments. Experiments with the bidirectional ring demonstrated differences for the other models when comparing values obtained at 50,000 bytes/sec to those obtained at 100,000 bytes/sec. The results obtained from the star and fully connected network topologies demonstrate differences in average response time values obtained with 50,000 and 100,000 bytes/sec bandwidths. This holds true for experiments conducted with all control models.

6.3 Work Requests Requiring Little Computation

6.3.1 The Environment

The third group of experiments was designed to demonstrate that differences in the performance of the control models do exist and that this difference can be observed when the time representing control overhead approaches the required service time for a work request. Each work request in this group of experiments named only an object file which performed a very short computation. No data file accesses were required. The probability that a work request arriving at any node named an object file residing on node i was 1/5 for i = 1 to 5.

In this set of experiments the following three factors were varied: 1) control model, 2) network topology, and 3) communication link bandwidth. The values used in this group of experiments are presented in Table 14. Experiments employing all possible combinations of these factors were conduc-

ted.

Table 14. Variables for the Group 3 Experiments

Control Models

XFDPS.1
XFDPS.2
XFDPS.3
XFDPS.4

Network Topology

Unidirectional Ring
Bidirectional Ring
Star
Fully Connected
Tree

Communication Link Bandwidth

1,200 bytes/sec
50,000 bytes/sec
100,000 bytes/sec
500,000 bytes/sec

6.3.2 Observations

The values for average response times obtained in the third group of experiments are presented in Table 15. A graphical representation of this data separated on the basis of network topology is given in Figures 52 through 56. A comparison of the average response time values obtained using the same network topology and communication bandwidth but different control models is provided in Table 16. A comparison of the values obtained with the same network topology and control model but different bandwidth is provided in Table 17.

In this group of experiments, a pattern is observed for the ordering of control models based on the average response times obtained utilizing these models. The following ordering is observed:

XFDPS.4 > XFDPS.1 > XFDPS.3 > XFDPS.2

This ordering is typically observed in experiments utilizing bandwidths of 1200 bytes/sec for all network topologies. In experiments using the higher bandwidths, the distinction between models XFDPS.1, XFDPS.2, and XFDPS.3 disappear, but the average response times obtained with XFDPS.4 remain significantly larger than those obtained with the other models.

When comparing the values for average response times obtained from experiments conducted with the unidirectional ring, bidirectional ring, and star network topologies using the same control model but varying the communication bandwidth, one observes changes at all bandwidths. Experiments utilizing the fully connected and tree network topologies provide changes at all bandwidths for only XFDPS.4. All other models demonstrate variances only at a bandwidth of 1200 bytes/sec.

6.4 Mixed Population of Work Requests

6.4.1 The Environment

The behavior of average response time for different types of jobs when the ratio of jobs is varied is investigated in the fourth group of experiments. The two types of work requests utilized in this set of experiments will be referred to as type 1 and 2 respectively. Type 1 work requests are identical to those used in the third group of experiments. They are characterized by accessing no data files and requiring very little processing time to complete. The object files named in the work requests are spread

Table 15. Average Work Request Response Time for Group 3

Unidirectional Ring

Bandwidth	XFDPS.1	XFDPS.2	XFDPS.3	XFDPS.4
1,200	4.9	0.6	3.4	2.1
50,000	0.041	0.038	0.039	0.047
100,000	0.033	0.031	0.032	0.037
500,000	0.027	0.025	0.026	0.031

Bidirectional Ring

Bandwidth	XFDPS.1	XFDPS.2	XFDPS.3	XFDPS.4
1,200	0.59	0.37	0.49	2.04
50,000	0.032	0.030	0.031	0.046
100,000	0.028	0.026	0.027	0.036
500,000	0.024	0.022	0.023	0.030

Star

Bandwidth	XFDPS.1	XFDPS.2	XFDPS.3	XFDPS.4
1,200	0.86	0.29	0.69	3.21
50,000	0.034	0.027	0.032	0.058
100,000	0.029	0.024	0.027	0.045
500,000	0.025	0.021	0.024	0.034

Fully Connected

Bandwidth	XFDPS.1	XFDPS.2	XFDPS.3	XFDPS.4
1,200	0.26	0.26	0.23	1.96
50,000	0.026	0.026	0.026	0.044
100,000	0.024	0.023	0.024	0.035
500,000	0.022	0.021	0.022	0.030

Tree

Bandwidth	XFDPS.1	XFDPS.2	XFDPS.3	XFDPS.4
1,200	1.12	0.36	0.85	4.01
50,000	0.035	0.030	0.033	0.069
100,000	0.029	0.026	0.029	0.051
500,000	0.025	0.022	0.025	0.038

Note: all values are in seconds

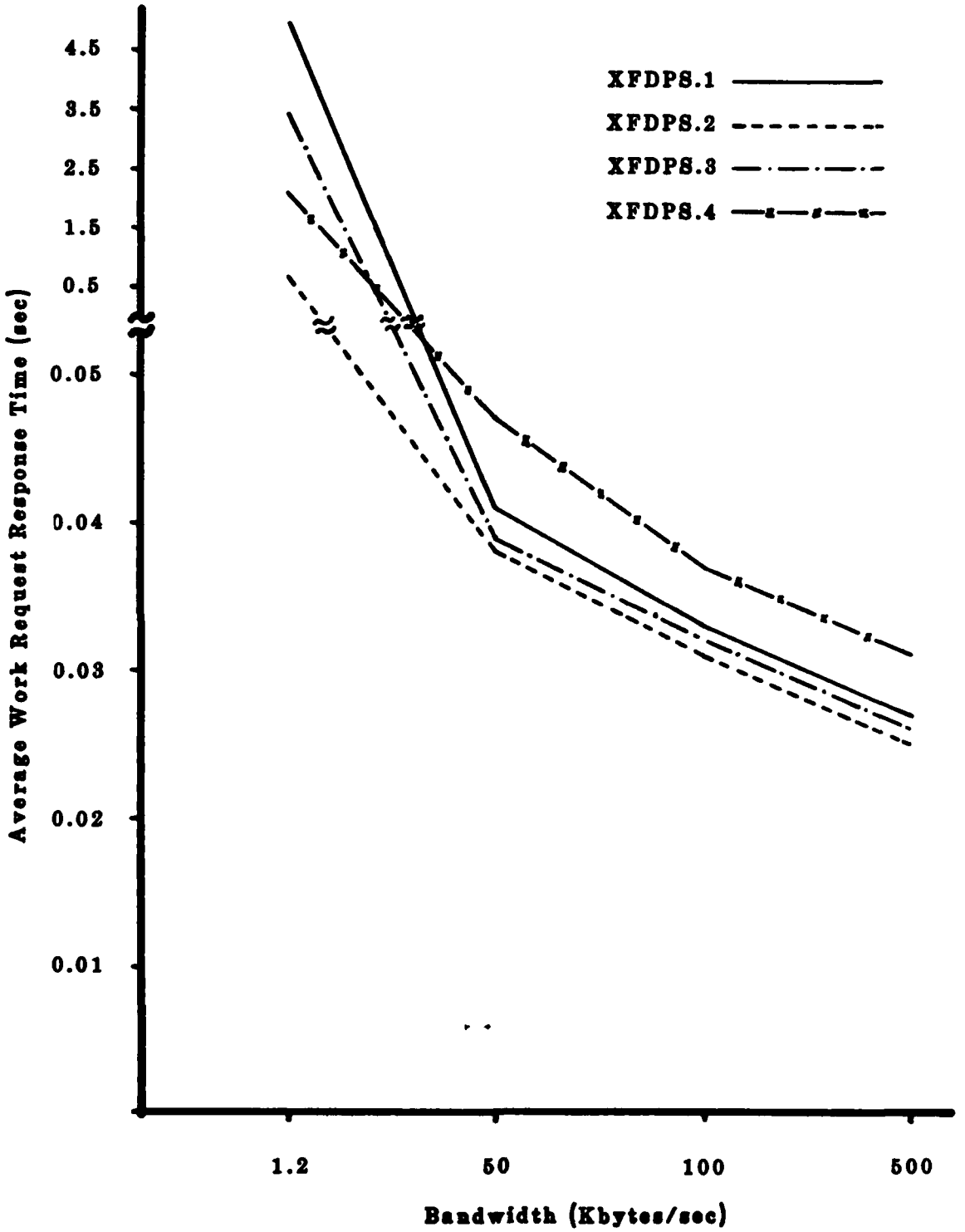


Figure 52. Average Work Request Response Time vs. Bandwidth for a Unidirectional Ring Network Topology for Group 3 Experiments

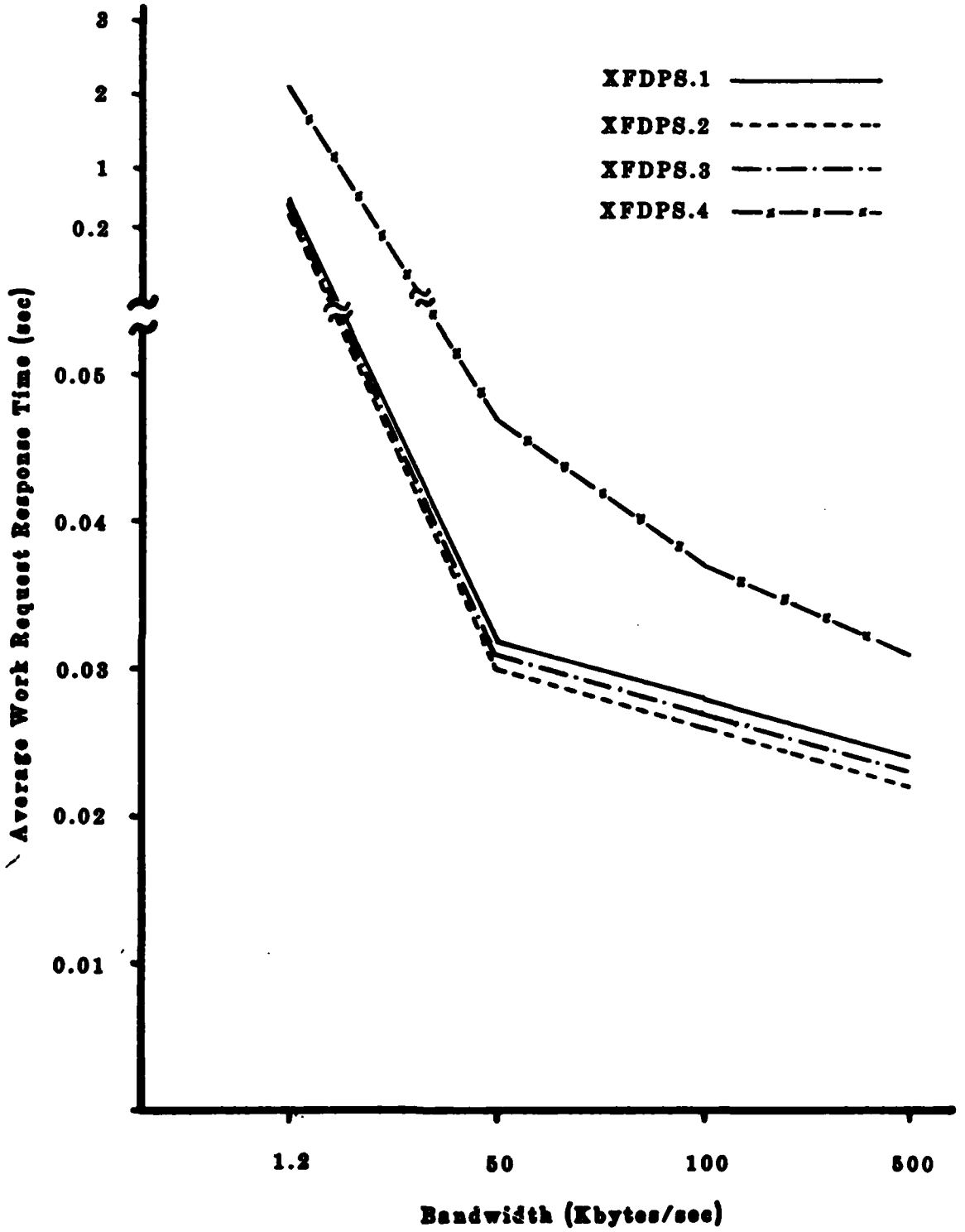


Figure 53. Average Work Request Response Time vs. Bandwidth for a Bidirectional Ring Network Topology for Group 3 Experiments

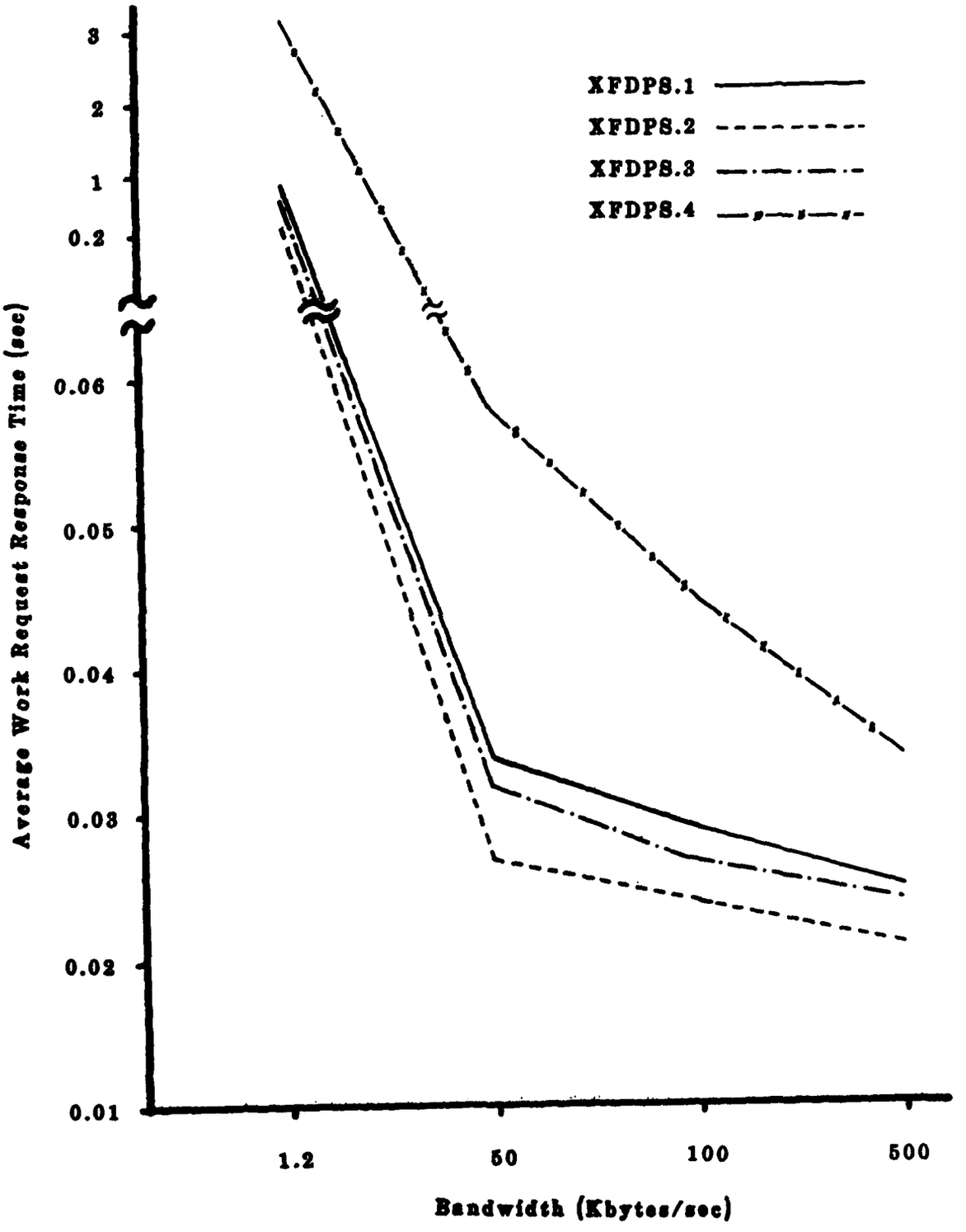


Figure 54. Average Work Request Response Time vs. Bandwidth
for a Star Network Topology
for Group 3 Experiments

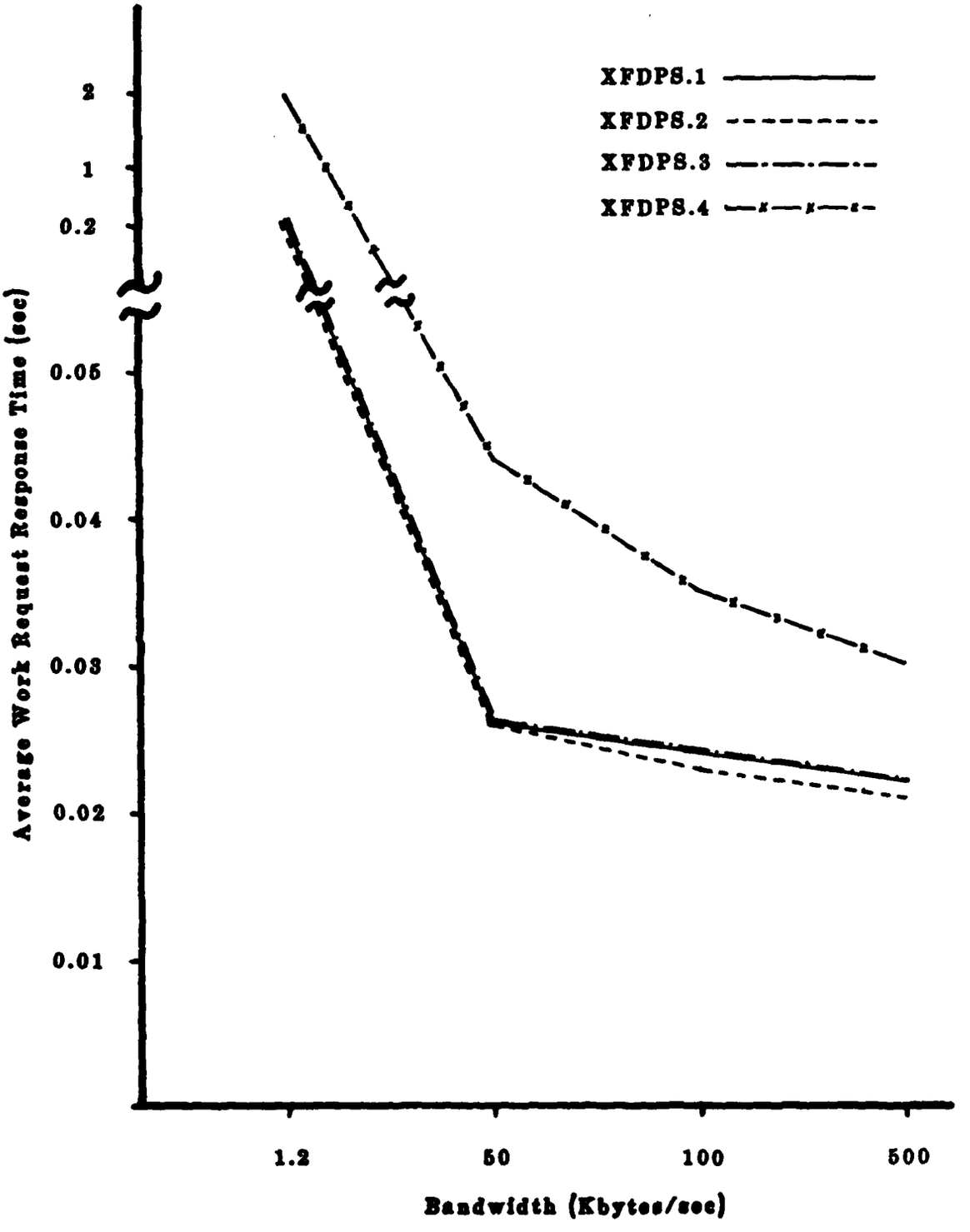


Figure 55. Average Work Request Response Time vs. Bandwidth for a Fully Connected Network Topology for Group 3 Experiments

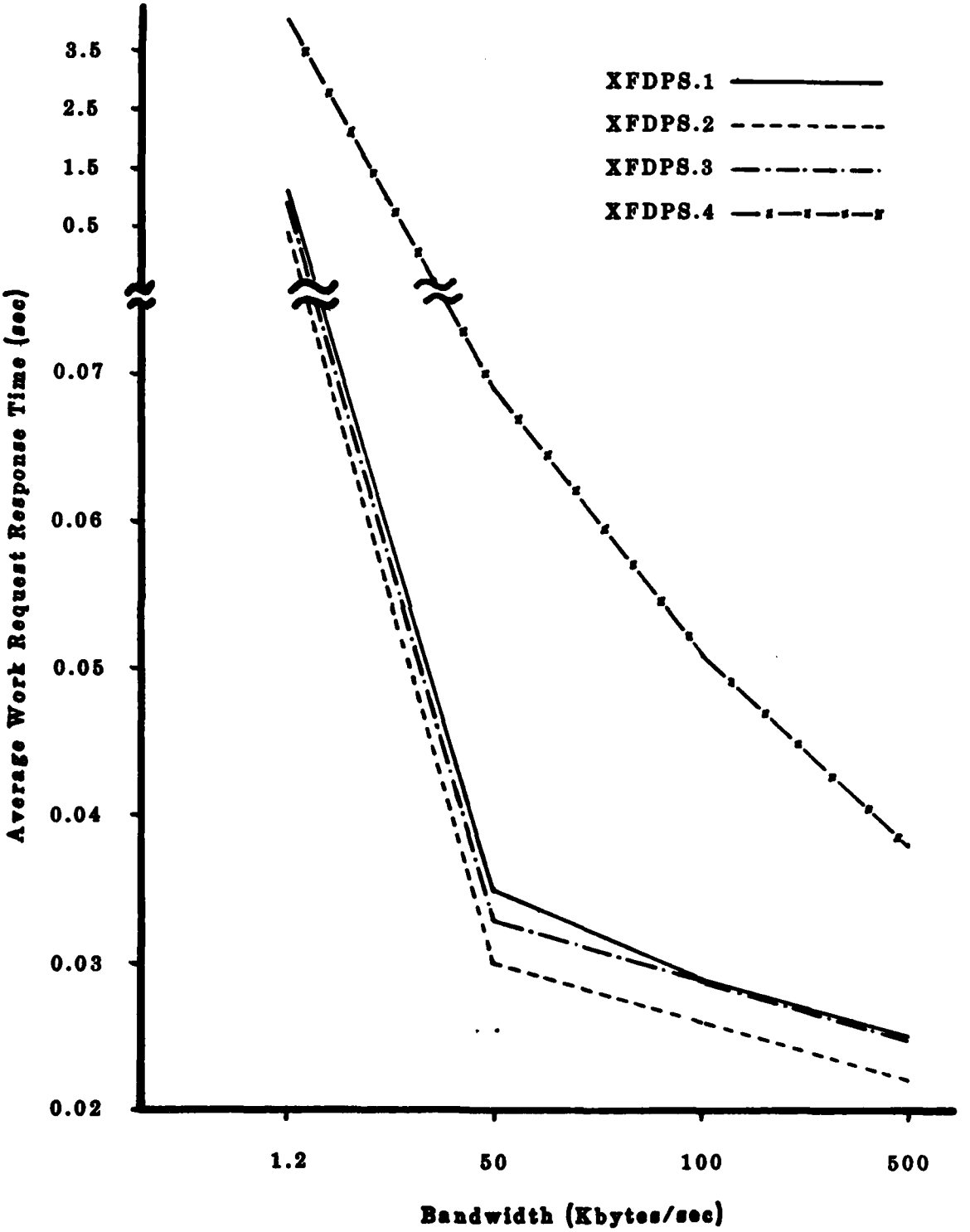


Figure 56. Average Work Request Response Time vs. Bandwidth
for a Tree Network Topology
for Group 3 Experiments

Table 16. Comparison of Response Times from Group 3 Experiments Using Different Control Models

Absolute Differences

Unidirectional Ring

Bandwidth	d12	d13	d14	d23	d24	d34
1,200	4.300	1.500	2.800	2.800	1.500	1.300
50,000	0.003	0.002	0.006	0.001	0.009	0.008
100,000	0.002	0.001	0.004	0.001	0.006	0.005
500,000	0.002	0.001	0.004	0.001	0.006	0.005

Bidirectional Ring

Bandwidth	d12	d13	d14	d23	d24	d34
1,200	0.220	0.100	1.450	0.120	1.670	1.550
50,000	0.002	0.001	0.014	0.001	0.016	0.015
100,000	0.002	0.001	0.008	0.001	0.010	0.009
500,000	0.002	0.001	0.006	0.001	0.008	0.007

Star

Bandwidth	d12	d13	d14	d23	d24	d34
1,200	0.570	0.170	2.350	0.400	2.920	2.520
50,000	0.007	0.002	0.024	0.005	0.031	0.026
100,000	0.005	0.002	0.016	0.003	0.021	0.018
500,000	0.004	0.001	0.009	0.003	0.013	0.010

Fully Connected

Bandwidth	d12	d13	d14	d23	d24	d34
1,200	0.000	0.030	1.700	0.030	1.700	1.730
50,000	0.000	0.000	0.018	0.000	0.018	0.018
100,000	0.001	0.000	0.011	0.001	0.012	0.011
500,000	0.001	0.000	0.008	0.001	0.009	0.008

Notation: dij = |RTi - RTj|, where RTi = Response time using XFDPS.i

(continued on next page)

Table 16. Comparison of Response Times from Group 3 Experiments Using Different Control Models (continued)

Absolute Differences

Tree

Bandwidth	d12	d13	d14	d23	d24	d34
1,200	0.760	0.270	2.890	0.490	3.650	3.160
50,000	0.005	0.002	0.034	0.003	0.039	0.036
100,000	0.003	0.000	0.022	0.003	0.025	0.022
500,000	0.003	0.000	0.013	0.003	0.016	0.013

Notation: dij = |RTi - RTj|, where RTi = Response time using XFDPS.i

Relative Differences

Unidirectional Ring

Bandwidth	d12	d13	d14	d23	d24	d34
1,200	0.88	0.31	0.57	0.82	0.71	0.38
50,000	0.07	0.05	0.13	0.03	0.19	0.17
100,000	0.06	0.03	0.11	0.03	0.16	0.14
500,000	0.07	0.04	0.13	0.04	0.19	0.16

Bidirectional Ring

Bandwidth	d12	d13	d14	d23	d24	d34
1,200	0.37	0.17	0.71	0.24	0.82	0.76
50,000	0.06	0.03	0.30	0.03	0.35	0.33
100,000	0.07	0.04	0.22	0.04	0.28	0.25
500,000	0.08	0.04	0.20	0.04	0.27	0.23

Notation: dij = |RTi - RTj| / Max (RTi, RTj)

(continued on next page)

Table 16. Comparison of Response Times from Group 3 Experiments Using Different Control Models (continued)

Relative Differences

Star

Bandwidth	d12	d13	d14	d23	d24	d34
1,200	0.66	0.20	0.73	0.58	0.91	0.79
50,000	0.21	0.06	0.41	0.16	0.53	0.45
100,000	0.17	0.07	0.36	0.11	0.47	0.40
500,000	0.16	0.04	0.26	0.13	0.38	0.29

Fully Connected

Bandwidth	d12	d13	d14	d23	d24	d34
1,200	0.00	0.12	0.87	0.12	0.87	0.88
50,000	0.00	0.00	0.41	0.00	0.41	0.41
100,000	0.04	0.00	0.31	0.04	0.34	0.31
500,000	0.05	0.00	0.27	0.05	0.30	0.27

Tree

Bandwidth	d12	d13	d14	d23	d24	d34
1,200	0.68	0.24	0.72	0.58	0.91	0.79
50,000	0.14	0.06	0.49	0.09	0.57	0.52
100,000	0.10	0.00	0.43	0.10	0.49	0.43
500,000	0.12	0.00	0.34	0.12	0.42	0.34

Notation: $dij = |RTi - RTj| / Max(RTi, RTj)$

Table 17. Comparison of Response Times from Group 3 Experiments Using Different Bandwidths but the Same Control Model

Absolute Differences

Unidirectional Ring

Bandwidths a	b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
1,200	50,000	4.9	0.6	3.4	2.1
50,000	100,000	0.0	0.0	0.0	0.0
100,000	500,000	0.0	0.0	0.0	0.0

Bidirectional Ring

Bandwidths a	b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
1,200	50,000	0.6	0.3	0.5	2.0
50,000	100,000	0.0	0.0	0.0	0.0
100,000	500,000	0.0	0.0	0.0	0.0

Star

Bandwidths a	b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
1,200	50,000	0.8	0.3	0.7	3.2
50,000	100,000	0.0	0.0	0.0	0.0
100,000	500,000	0.0	0.0	0.0	0.0

Fully Connected

Bandwidths a	b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
1,200	50,000	0.2	0.2	0.2	1.9
50,000	100,000	0.0	0.0	0.0	0.0
100,000	500,000	0.0	0.0	0.0	0.0

(continued on next page)

Table 17. Comparison of Response Times from Group 1 Experiments Using Different Control Models but the Same Bandwidth (continued)

Absolute Differences

Tree

Bandwidths a	b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
1,200	50,000	1.1	0.3	0.8	3.9
50,000	100,000	0.0	0.0	0.0	0.0
100,000	500,000	0.0	0.0	0.0	0.0

Relative Differences

Unidirectional Ring

Bandwidths a	b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
1,200	50,000	0.99	0.94	0.99	0.98
50,000	100,000	0.20	0.18	0.18	0.21
100,000	500,000	0.18	0.19	0.19	0.16

Bidirectional Ring

Bandwidths a	b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
1,200	50,000	0.95	0.92	0.94	0.98
50,000	100,000	0.13	0.13	0.13	0.22
100,000	500,000	0.14	0.15	0.15	0.17

Star

Bandwidths a	b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
1,200	50,000	0.96	0.91	0.95	0.98
50,000	100,000	0.15	0.11	0.16	0.22
100,000	500,000	0.14	0.13	0.11	0.24

(continued on next page)

Table 17. Comparison of Response Times from Group 1 Experiments Using Different Control Models but the Same Bandwidth (continued)

Relative Differences

Fully Connected

Bandwidths a	b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
1,200	50,000	0.90	0.90	0.89	0.98
50,000	100,000	0.08	0.12	0.08	0.20
100,000	500,000	0.08	0.09	0.08	0.14

Tree

Bandwidths a	b	XFDPS.1 dab	XFDPS.2 dab	XFDPS.3 dab	XFDPS.4 dab
1,200	50,000	0.97	0.92	0.96	0.98
50,000	100,000	0.17	0.13	0.12	0.26
100,000	500,000	0.14	0.15	0.14	0.25

evenly across all nodes of the network.

The type 2 work requests are identical to those used in the second group of experiments. Object-data file pairs are named in the work requests. In each case the object file and data file reside on different nodes. If the object file resides on node i, the data file resides on node j where

$$j = \begin{cases} i + 1, & i < 5 \\ 1, & i = 5 \end{cases}$$

The object-data file pairs are spread evenly across all nodes.

In this set of experiments the following two sets of factors are varied: control model and ratio of type 1 and type 2 jobs. In all experiments a unidirectional ring network topology with a communication bandwidth of 50,000 bytes per second is utilized. The control models used in these experiments

are XFDPS.1, XFDPS.2, XFDPS.3, and XFDPS.4. The ratio of type 1 to type 2 work requests in the population of work requests is initialized at the following different values: 10%, 50%, and 90% type 2 work requests. The actual ratio observed in each experiment is reported in the results described below.

6.4.2 Observations

Table 18 contains the average work request response times for the fourth group of experiments. A plot of average work request response time for type 1 work requests versus the fraction of work requests which were type 2 work requests is provided in Figure 57. A similar plot for type 2 work requests is provided in Figure 58, and one for all work requests is found in Figure 59.

The impact of an increase in communication traffic as a result of increasing the frequency of type 2 work requests on the average work request response time for type 1 jobs is observed. The increase, though, does not persist, and the average response times decline somewhat as the percentage of type 2 jobs approaches extremely high values. The values obtained with the different control models indicate that XFDPS.4 results are consistently higher than those of all other models. XFDPS.2 performs the best when the smallest percentage of type 2 work requests is present, but its performance degrades more than XFDPS.1 and XFDPS.2 as higher percentages of type 2 work requests are observed. The results using XFDPS.1 and XFDPS.3 do not differ very much.

The average response time for type 2 work requests also increases as the percentage of these work requests increases. The values for average response time, though, appear to remain relatively unchanged after the relative percentage of type 2 work requests reaches fifty percent. The average response times obtained with XFDPS.2 are less than those obtained with the other models when the percentage of type 2 work requests is the smallest, but the differences decrease significantly as the job mix is increased in favor of type 2 work requests.

When both types of jobs are considered together, an increase in average work request response time is observed as the frequency of type 2 work requests is increased. Differences among the results obtained with the different models is not observed.

Table 18. Average Work Request Response Time for Group 4

XFDPS.1

Predicted* Job Mix	Actual Job Mix	Average Work Request Response Time (sec) Type 1**	Type 2***	All
0	0	0.041	-----	0.041
10	8.82	0.450	99.1	9.15
50	46.6	0.708	148.5	69.6
90	94.3	0.587	157.7	148.8

XFDPS.2

Predicted* Job Mix	Actual Job Mix	Average Work Request Response Time (sec) Type 1**	Type 2***	All
0	0	0.038	-----	0.038
10	11.7	0.233	55.5	6.70
50	46.5	0.875	150.0	70.3
90	92.3	0.681	147.7	136.3

XFDPS.3

Predicted* Job Mix	Actual Job Mix	Average Work Request Response Time (sec) Type 1**	Type 2***	All
0	0	0.039	-----	0.039
10	8.73	0.422	97.1	8.87
50	51.4	0.724	172.1	88.8
90	90.4	0.665	171.9	155.4

XFDPS.4

Predicted* Job Mix	Actual Job Mix	Average Work Request Response Time (sec) Type 1**	Type 2***	All
0	0	0.047	-----	0.047
10	9.34	1.08	90.1	9.39
50	48.4	1.38	161.0	78.6
90	94.1	1.19	147.9	139.3

* Percent of all work requests that are type 2 work requests
** Type 1 work requests compute for a short interval and access no files
*** Type 2 work requests compute for a relatively long interval and access files residing on distant nodes

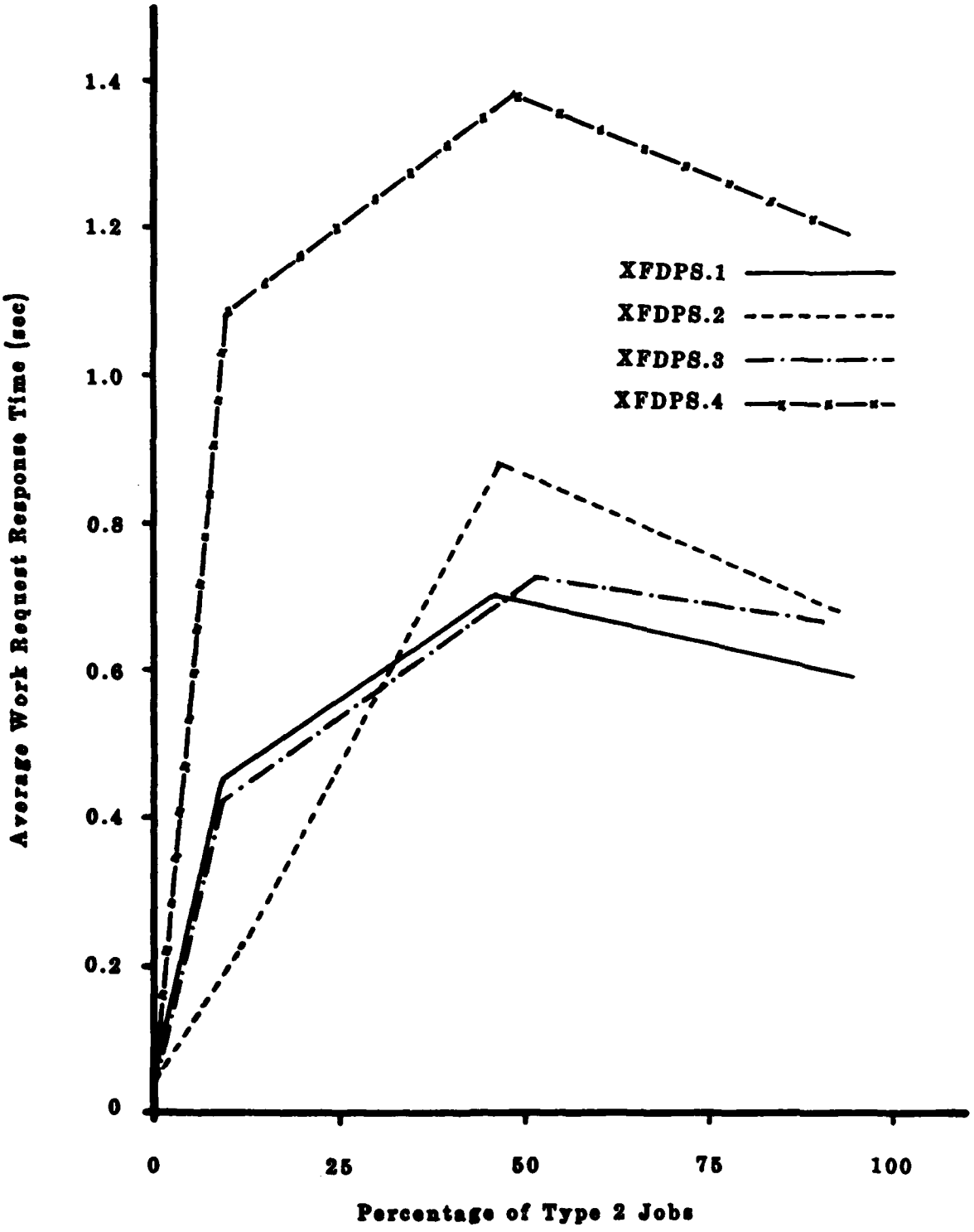


Figure 57. Average Work Request Response Time vs. Job Mix for Type 1 Jobs in the Group 4 Experiments

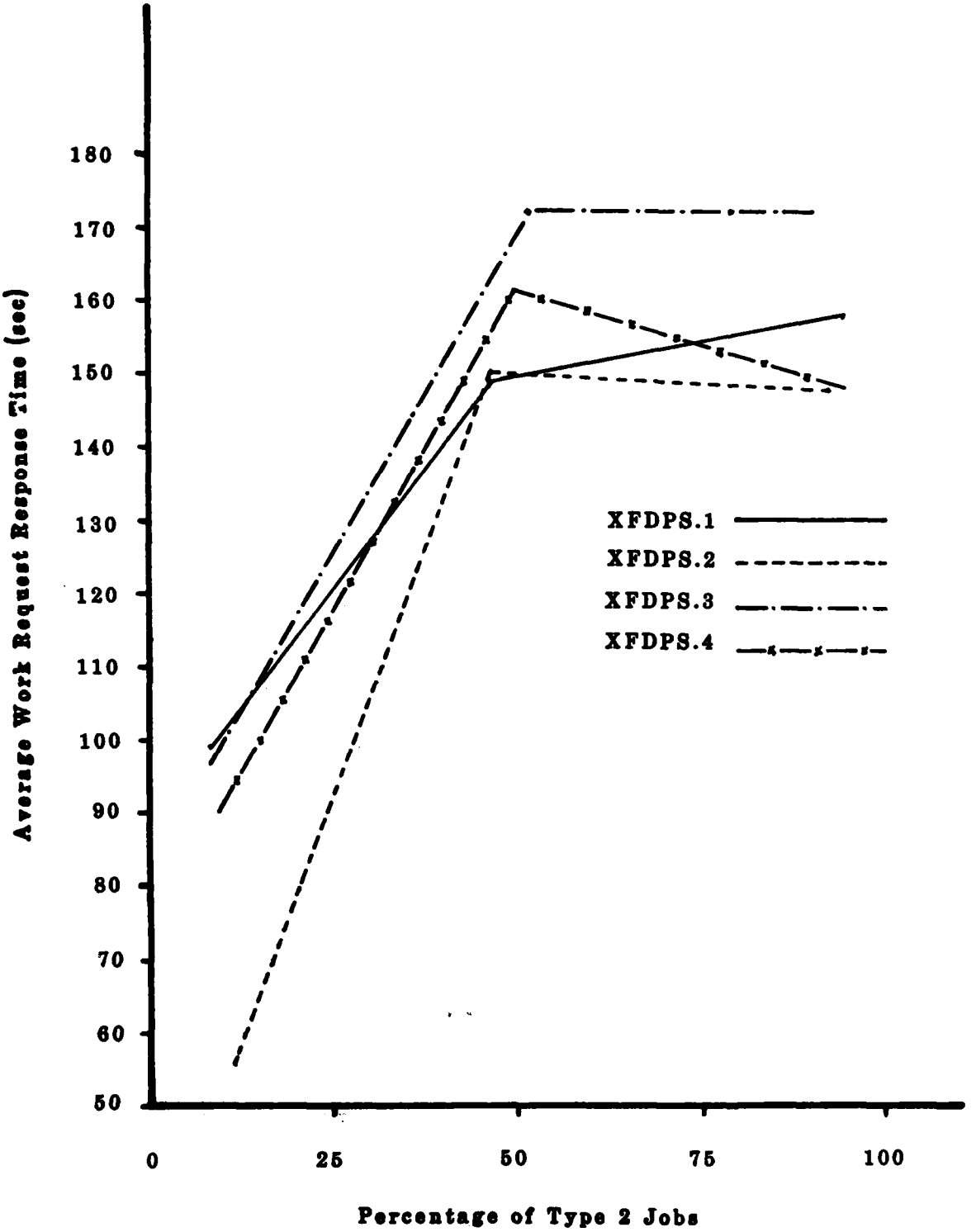


Figure 58. Average Work Request Response Time vs. Job Mix for Type 2 Jobs in the Group 4 Experiments

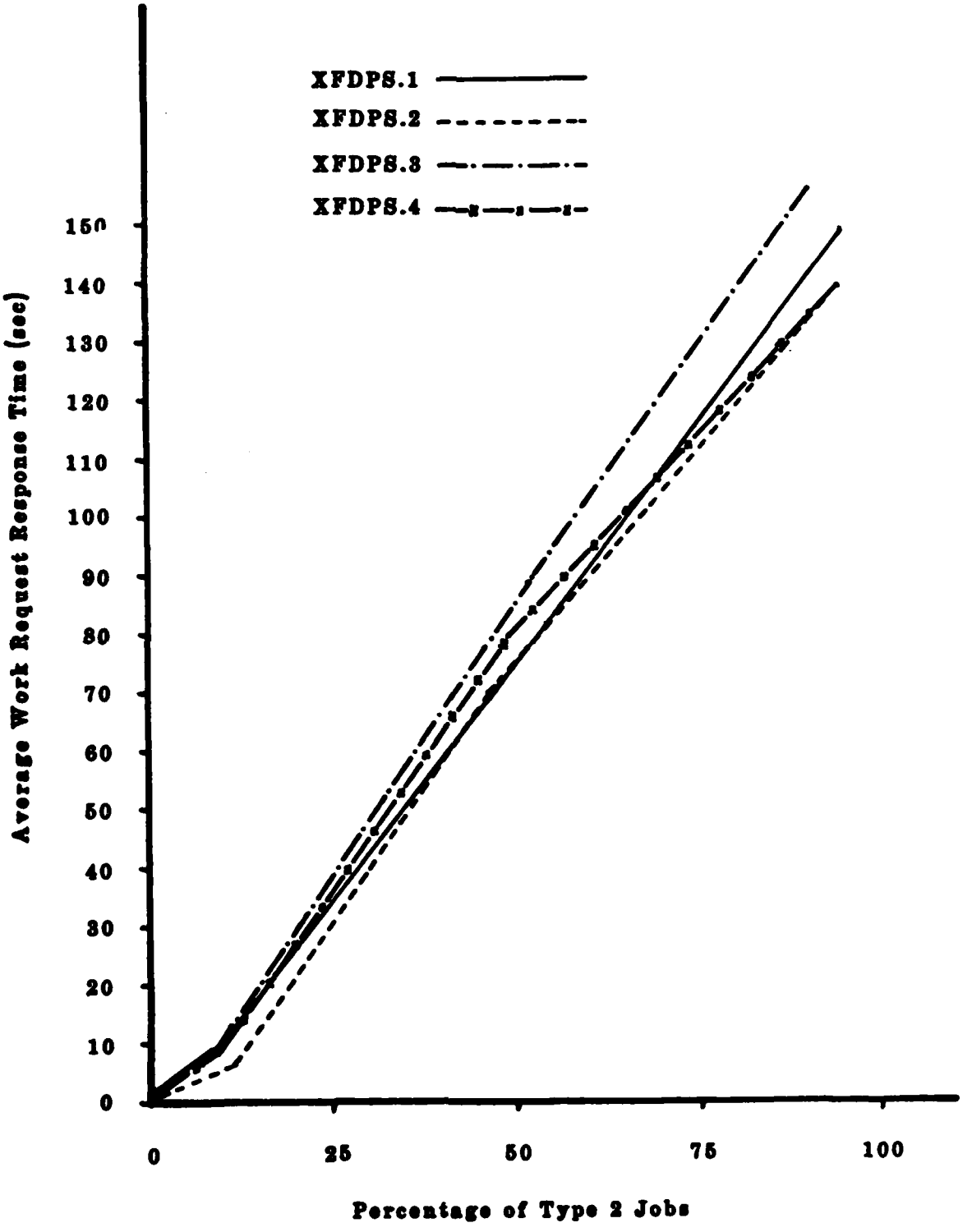


Figure 59. Average Work Request Response Time vs. Job Mix for All Jobs in the Group 4 Experiments

6.5 Simulation of a One Node Network

6.5.1 The Environment

This set of experiments was considered separately from the four groups of experiments described above because its purpose was not to analyze the relative performance of the control models. These experiments were designed to provide a standard upon which the other results could be compared in order to determine the impact of distributed processing on average response time for work requests.

In this set of experiments, the network consisted of only a single node. This single node was identical to the nodes used in the other experiments. The work requests named object-data file pairs in which the script for the object file was the same as that employed in the first two groups of experiments. Since there was no internode communication, the choice of the control model was of no consequence, and therefore XFDPS.1 was arbitrarily selected.

6.5.2 Observations

Five simulations were conducted and the results of those runs are presented in Table 19. The values for average response time from these experiments are similar to those found in the first group of experiments when bandwidths greater than 600 bytes/sec are used (see Table 7).

Table 19. Average Work Request Response Time for a Single Node Network

Run	Average Response Time (sec)
1	44.6
2	44.1
3	43.7
4	43.7
5	44.2

Mean: 44.1 seconds

Standard Deviation: 0.38

SECTION 7

ANALYSIS OF THE SIMULATION EXPERIMENTS

In this chapter, the data obtained from the simulation experiments is analyzed. The first section presents an analysis of the results of simulation experiments involving a single node network. The second section analyzes the three groups of experiments involving the simulation of an FDPS environment.

Analysis of the simulation data would be incomplete without a statement as to the validity of the results as predictors of results obtained from real systems. In the analysis of the single node network experiments, the simulation data is compared to data obtained from an analytical model which has been established as a good predictor by comparison with data obtained from actual running systems. The results of this comparison indicate that the simulation data is quite similar to the analytically obtained data.

Validation of the simulation data obtained from experiments with five node networks cannot be accomplished by the same means because analytical models have not yet been developed and there does not exist running systems from which real data can be obtained. Therefore, the simulation data cannot be validated, but confidence in the predictability of the trends in the behavior of the control models can still be developed by comparing the conclusions obtained with this simulation data to the conclusions made by other experimenters using simulation techniques.

7.1 Single Node Network Experiments

The single node network simulated in this set of experiments can be considered to be a simple timesharing system. Timesharing systems have been extensively studied resulting in the construction of analytical models describing these systems.

Figure 60 depicts a model of a timesharing system. There is a fixed number of user terminals M that are serviced by a single server S. Work from the terminals is fed into a single queue that is serviced by a round robin policy. "Think time," the average delay between work requests from a given terminal, is assumed to be exponentially distributed with a mean of 1/L. The average service time for each work request is also assumed to be exponentially

distributed with mean of 1/u. The probability of finding m terminals actively competing for the server S is denoted by P_m. The average response time for a work request is denoted by T. Kleinrock [Klei68] gives the following result for computing T:

$$T = \frac{M}{u * (1 - P0)} - \frac{1}{L}$$

$$\text{where } P0 = \left[\sum_{m=0}^{M} \frac{M!}{(M-m)!} * \left(\frac{L}{u}\right)^m \right]^{-1}$$

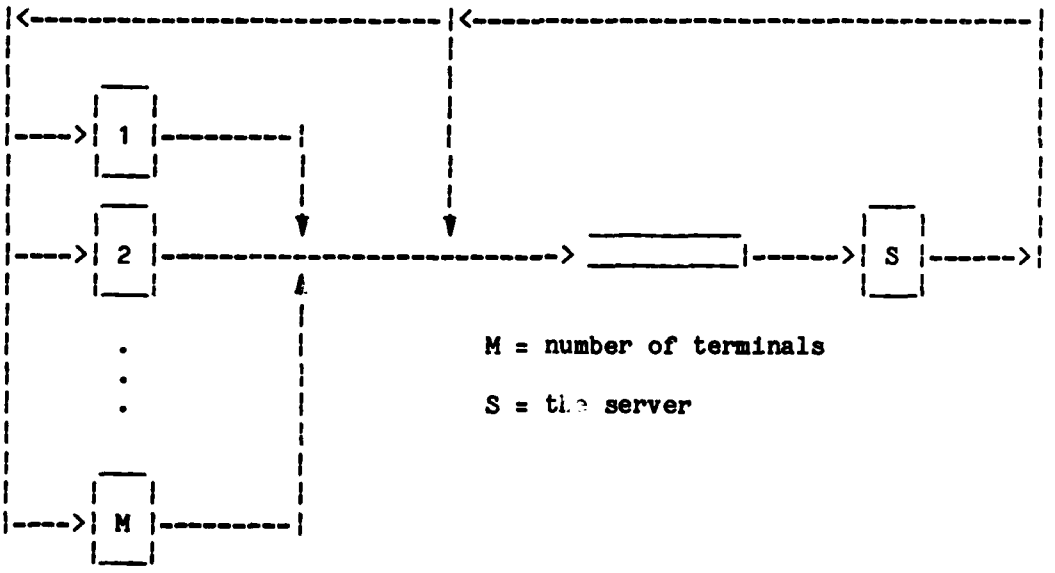


Figure 60. Model of a Timesharing System

This result can be used to compare analytically predicted average response times with values obtained by means of simulation. Assuming ten terminals (M = 10) and an average service time of 5.01 seconds (1/u = 5.01),

analytically predicted values for a system possessing the characteristics simulated in the single node network experiments can be computed. Table 20 contains the analytically computed values for average work request response time.

Table 20. Analytically Computed Values for the Average Response Time in a Timesharing System

Ave. Think Time (sec)	Ave. Response Time (sec)
1	49.1
2	48.1
3	47.1
4	46.1
5	45.1
6	44.1
7	43.1
8	42.1
9	41.1
10	40.1
11	39.1
12	38.1
13	37.1
14	36.1
15	35.1

The mean value for the average work request response time from the five simulation runs utilizing a single node network is 44.1 seconds (see Table 19). This value corresponds to the analytically computed value using a "think time" of six seconds. This result increases one's confidence in the values produced by the simulator.

7.2 Five Node Network Experiments

The results of the first group of experiments indicate that there is no significant difference in the values for average work request response time obtained with the various control models utilizing communication systems with

bandwidths larger than 600 bytes/sec. The response time values obtained with communication systems using the higher bandwidths are similar to those obtained for a single node network. This indicates that the delay experienced in processing a work request in the first group of experiments can be explained by the queueing delays experienced in the process queues of the processor on which the work request is being serviced. There appears to be no measurable delay due to the actions of the FDPS executive control in providing a fully distributed environment.

Message traffic required for each work request by the various control models must be analyzed in order to see why the delay due to FDPS executive control actions is overshadowed by that experienced in simply executing the process named in the work request once it has been initiated. In this analysis only models XFDPS.1 and XFDPS.2 are compared. XFDPS.3 is similar to XFDPS.1 except that fewer messages are required in certain instances when resources are found to be local. XFDPS.4 does not lend itself to this type of analysis because work requests are treated in a batch when a node receives the control vector that provides access to the resource tables.

An analysis of the control messages required for each work request employing XFDPS.1 is provided in Table 21 and that for XFDPS.2 is provided in Table 22. Work requests from the first group of experiments require fifteen internodal messages under XFDPS.1 and nine under XFDPS.2. Therefore, six more control messages are required by XFDPS.1.

Average link queue waiting times for experiments conducted with XFDPS.1 and XFDPS.2 with all topologies and bandwidths ranging from 1200 to 500,000 bytes/sec can be found in Tables 23 and 24. These values are rather small and indicate that the communication system never seems to become a bottleneck at these bandwidths. Similar results have been reported in [Souz81]. All of the values for average link queue waiting time in these tables is less than or equal to 0.05 seconds. In order to perform a worst case analysis of the overhead due to the executive control message traffic, the following assumptions are made:

1. waiting time in each link queue is 0.05 seconds
2. control messages are 50 bytes long
3. the bandwidth of each link is 1200 bytes/sec

4. a message must traverse four links in order to reach its destination (this represents the longest path present in a system with a unidirectional ring network topology)

These assumptions imply that each message experiences a delay of 0.37 seconds as shown below:

```
message delay = total link queue delay +
                total message transmission time

              = [(4 link queues) *
                (0.05 sec/link queue)] +
                [(4 links) *
                (50 bytes / 1200 bytes/sec/link)]

              = 0.37 seconds
```

Assuming XFDPS.1 requires fifteen messages per work request, the total time for executive control message traffic is 5.5 seconds, or approximately twelve percent of the average work request response time (recall that this quantity is observed to be in the neighborhood of 45 seconds). This is a worst case analysis and consequently one would expect a much smaller fraction of the response time attributable to executive control traffic on the average due to a number of factors including the fact that not all messages must traverse four links, and in certain situations messages can be processed in parallel. Comparison of the simulation results demonstrates that a ten percent variation in the values for average response time does not represent a significant variation. Therefore, the time attributable to the executive control is not considered a significant factor in the value for average response time.

A similar calculation for XFDPS.2 results in a total time for executive control message traffic of 3.3 seconds, or approximately seven percent of the average work request response time. Again, the time that can be attributed to the transmission of executive control messages is not considered to be a significant contribution to the value of average response time.

The difference in message traffic between the two models is six messages. This results in an executive control message delay of 2.2 seconds or approximately five percent of the average work request response time. Therefore, no significant difference in executive control overhead is predicted for the processing of work requests from the group 1 experiments. This

Table 21. Control Messages Required for a Work Request Under XFDPS.1

Activity	Maximum Number of Internode Messages
request for resource availability info.	N - 1
resource availability info.	N - 1
file lock and release requests	A
results of the file locks and releases	A
process activation request	L
process termination notification	T
file release request	L
total = 2 * (N - 1) + (2 * A) + (2 * L) + T	

N: number of physical nodes in the network
L: number of files named in the work request
T: number of tasks in the work request
A: number of nodes possessing available resources which are required by the work request

For Group I Experiments:

N = 5
L = 2
T = 1
A = 1

total = 15

lack of variation is observed in the simulation results.

The results from the second group of simulation experiments can be analyzed by comparing the number of executive control messages (15 for XFDPS.1 and 9 for XFDPS.2) to the number of user messages (501 remote file accesses per work request). The fraction of message traffic for each work request that can be attributed to the FDPS executive control is approximately three percent for XFDPS.1 and two percent for XFDPS.2. The difference in message traffic between the two models is approximately one percent of the user message

Table 22. Control Messages Required for a Work Request Under XFDPS.2

Activity	Maximum Number of Internode Messages
request for resource availability info.	1
resource availability info.	1
file lock and release requests	1
results of the file locks and releases	1
process activation request	L
process termination notification	T
file release request	T
file process deactivation request	L - T
	total = 2 * L + T + 4

N: number of physical nodes in the network
L: number of files named in the work request
T: number of tasks in the work request
A: number of nodes possessing available resources which are required by the work request

For Group I Experiments:

N = 5
L = 2
T = 1
A = 1

total = 9

traffic required to perform remote file accesses. These values demonstrate that one should not observe a measurable difference in work request response times when employing the different models to process work requests of the type found in the second group of experiments.

The first two groups of experiments demonstrate a situation in which the resource demands required to service a work request once all of the initialization tasks have been accomplished by the executive control far

Table 23. Average Wait Time in the Link Queues for Group 1 Experiments Using XFDPS.1

Unidirectional Ring

Bandwidth	L11	L21	L31	L41	L51
1,200	0.034	0.043	0.054	0.038	0.031
50,000	0.0008	0.0009	0.0009	0.001	0.0008
100,000	0.0006	0.0006	0.0006	0.0006	0.0005
500,000	0.0006	0.0005	0.0005	0.0005	0.0005

Bidirectional Ring

Bandwidth	L11	L12	L21	L22	L31	L32
1,200	0.033	0.031	0.041	0.031	0.040	0.034
50,000	0.0010	0.0008	0.0011	0.0008	0.0011	0.0008
100,000	0.0006	0.0006	0.0008	0.0005	0.0007	0.0006
500,000	0.0004	0.0004	0.0006	0.0003	0.0005	0.0004

Bidirectional Ring

Bandwidth	L41	L42	L51	L52
1,200	0.037	0.030	0.038	0.040
50,000	0.0010	0.0011	0.0011	0.0011
100,000	0.0006	0.0007	0.0007	0.0007
500,000	0.0004	0.0004	0.0005	0.0004

Star

Bandwidth	L11	L12	L13	L14	L21	L31	L41	L51
1,200	0.03	0.03	0.04	0.03	0.07	0.07	0.07	0.07
50,000	0.0008	0.0008	0.0008	0.0007	0.002	0.002	0.002	0.002
100,000	0.0005	0.0005	0.0005	0.0005	0.0012	0.0012	0.0013	0.0012
500,000	0.0003	0.0003	0.0003	0.0003	0.0008	0.0008	0.0008	0.0007

Notation: Lij denotes the mean wait time for link queue j on node i

(continued on next page)

Table 23. Average Wait Time in the Link Queues for Group 1 Experiments Using XFDPS.1 (continued)

Fully Connected

Bandwidth	L11	L12	L13	L14	L21	L22	L23	L24
1,200	0.04	0.04	0.04	0.04	0.03	0.04	0.04	0.04
50,000	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
100,000	0.0006	0.0006	0.0007	0.0006	0.0006	0.0006	0.0006	0.0006
500,000	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003

Fully Connected

Bandwidth	L31	L32	L33	L34	L41	L42	L43	L44
1,200	0.04	0.04	0.04	0.04	0.03	0.04	0.03	0.03
50,000	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
100,000	0.0005	0.0007	0.0006	0.0005	0.0005	0.0006	0.0006	0.0006
500,000	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003

Fully Connected

Bandwidth	L51	L52	L53	L54
1,200	0.04	0.03	0.04	0.04
50,000	0.001	0.001	0.001	0.001
100,000	0.0006	0.0006	0.0006	0.0006
500,000	0.0003	0.0003	0.0003	0.0003

Tree

Bandwidth	L11	L12	L21	L22	L23	L31	L41	L51
1,200	0.04	0.02	0.04	0.03	0.03	0.07	0.07	0.07
50,000	0.0009	0.0009	0.0009	0.0005	0.0006	0.002	0.002	0.002
100,000	0.0005	0.0003	0.0006	0.0003	0.0004	0.001	0.001	0.001
500,000	0.0004	0.0002	0.0003	0.0002	0.0003	0.0008	0.0008	0.0007

Notation: Lij denotes the mean wait time for link queue j on node i

outweigh the resource demands by the executive control required to perform the initialization tasks. The third group of simulation experiments demonstrates

Table 24. Average Wait Time in the Link Queues for Group 1 Experiments Using XFDPS.2

Unidirectional Ring

Bandwidth	L11	L21	L31	L41	L51
1,200	0.04	0.02	0.02	0.02	0.02
50,000	0.001	0.0003	0.0006	0.0007	0.0007
100,000	0.0007	0.0002	0.0003	0.0004	0.0005
500,000	0.0003	0.0001	0.0002	0.0002	0.0002

Bidirectional Ring

Bandwidth	L11	L12	L21	L22	L31	L32
1,200	0.04	0.04	0.01	0.02	0.03	0.03
50,000	0.001	0.001	0.0004	0.0003	0.001	0.0007
100,000	0.0007	0.0007	0.0002	0.0002	0.0005	0.0004
500,000	0.0003	0.0003	0.0002	0.0002	0.0002	0.0003

Bidirectional Ring

Bandwidth	L41	L42	L51	L52
1,200	0.03	0.03	0.02	0.02
50,000	0.0007	0.0009	0.0004	0.0004
100,000	0.0004	0.0005	0.0003	0.0003
500,000	0.0002	0.0003	0.0002	0.0002

Star

Bandwidth	L11	L12	L13	L14	L21	L31	L41	L51
1,200	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03
50,000	0.0009	0.0009	0.0008	0.0009	0.0006	0.0007	0.0007	0.0007
100,000	0.0005	0.0005	0.0005	0.0005	0.0004	0.0003	0.0004	0.0005
500,000	0.0002	0.0002	0.0003	0.0002	0.0003	0.0003	0.0003	0.0003

Notation: Lij denotes the mean wait time for link queue j on node i

(continued on next page)

Table 24. Average Wait Time in the Link Queues for Group 1 Experiments Using XFDPS.2 (continued)

Fully Connected

Bandwidth	L11	L12	L13	L14	L21	L22	L23	L24
1,200	0.04	0.04	0.04	0.04	0.02	0.04	0.04	0.04
50,000	0.001	0.001	0.001	0.001	0.0003	0.001	0.001	0.001
100,000	0.0007	0.0007	0.0007	0.0007	0.0002	0.0007	0.0007	0.0007
500,000	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003

Fully Connected

Bandwidth	L31	L32	L33	L34	L41	L42	L43	L44
1,200	0.02	0.04	0.05	0.04	0.02	0.04	0.04	0.04
50,000	0.0003	0.001	0.001	0.001	0.0003	0.001	0.001	0.001
100,000	0.0002	0.0007	0.0007	0.0007	0.0002	0.0007	0.0007	0.0007
500,000	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003

Fully Connected

Bandwidth	L51	L52	L53	L54
1,200	0.02	0.04	0.04	0.04
50,000	0.0002	0.001	0.001	0.001
100,000	0.0002	0.0007	0.0007	0.0007
500,000	0.0003	0.0003	0.0003	0.0003

Tree

Bandwidth	L11	L12	L21	L22	L23	L31	L41	L51
1,200	0.04	0.04	0.02	0.01	0.009	0.02	0.03	0.03
50,000	0.001	0.001	0.0003	0.0002	0.0003	0.0006	0.001	0.001
100,000	0.0007	0.0006	0.0002	0.0002	0.0002	0.0004	0.0007	0.0007
500,000	0.0003	0.0003	0.0002	0.0001	0.0001	0.0003	0.0003	0.0003

Notation: Lij denotes the mean wait time for link queue j on node i

that as the service time requirements of the work request decrease and approach the requirements of the executive control, differences among the per-

formance of the models will appear. Since XFDPS.2 requires the transmission of fewer messages than XFDPS.1, it is predicted to demonstrate better performance. This is observed in the results of the simulation experiments. It is also observed that XFDPS.3 has a performance that places it between that of XFDPS.2 and XFDPS.1. This is to be expected because XFDPS.3 functions in an identical manner to XFDPS.1 when all resources cannot be found locally but experiences much less overhead if resources are found to all reside locally. In group 3 experiments the probability of all resources being found locally is 1/5.

The performance of XFDPS.4 is found to be consistently inferior to that of the other models when processing the work requests from the third group of experiments. This can be attributed to the lack of parallel activity in the management of resources. Only one node at a time is permitted to allocate and deallocate resources. The next node to receive allocation and deallocation permission is delayed in performing these actions until all nodes receive the updates to the resource directory by the previous holder of the allocate and deallocate privileges. When the service times of the work requests are long, this initial delay is insignificant. Results from the group 3 experiments demonstrate that the impact on jobs with small service times is quite significant.

The fourth group of experiments demonstrates the impact on time delays attributable to control overhead as a result of an increase in communication traffic. In this set of experiments the mixture of two types of jobs is varied. The third group of experiments represents the situation in which there are only type 1 jobs present, and the second group of experiments represents the situation in which there are only type 2 jobs present. As the fraction of type 2 jobs is increased, the average work request response time for type 1 jobs increases. Type 1 jobs are given special attention because they were observed to be sensitive to changes in control overhead in the third group of experiments. The observed increase in message delays is presented in Table 25. It is the increase in these delays that are responsible for the observed increase in control overhead.

It is observed in the fourth group of experiments that the average response times for type 1 work requests decreases when the percentage of type

2 work requests is increased from around fifty percent to ninety percent. Type 2 work requests name object files which execute a long sequence of compute instructions followed by remote file accesses. When these processes attempt a file access, they are blocked until the access operation is completed thus releasing the processor to other processes. As the frequency of this type of job is increased, the average delay in the blocked state increases. Therefore, processes resulting from type 1 work requests experience fewer processes waiting in the ready state for the processor as the number of type 2 work requests is increased. This means the queueing delay experienced by type 1 work requests should decrease.

Table 25. Average Wait Time in the Link Queues for Group 4 Experiments

XFDPS.1

Job Mix*	L11	L21	L31	L41	L51	Max
0	0.0010	0.0009	0.0010	0.0011	0.0010	0.0011
8.82	0.0572	0.0483	0.0496	0.0050	0.0453	0.0572
46.6	0.0933	0.0071	0.1664	0.0562	0.0153	0.1664
94.3	0.0112	0.1385	0.1509	0.0066	0.0080	0.1385
100	0.5656	0.2290	0.0212	0.0252	0.0285	0.5656

XFDPS.2

Job Mix*	L11	L21	L31	L41	L51	Max
0	0.0003	0.0004	0.0004	0.0003	0.0003	0.0004
11.7	0.0260	0.0269	0.0292	0.0051	0.0207	0.0292
46.5	0.1284	0.1464	0.0208	0.0046	0.0203	0.1464
92.3	0.0716	0.0858	0.0102	0.0690	0.0714	0.0858
100	0.0118	0.0241	0.6039	0.2248	0.0135	0.6039

XFDPS.3

Job Mix*	L11	L21	L31	L41	L51	Max
0	0.0010	0.0008	0.0008	0.0009	0.0009	0.0010
8.73	0.0206	0.0603	0.0698	0.0059	0.0524	0.0698
51.4	0.1000	0.1382	0.0255	0.0048	0.1437	0.1437
90.4	0.0067	0.1292	0.0694	0.1780	0.0065	0.1780
100	0.0966	0.0973	0.5833	0.0778	0.0083	0.5833

XFDPS.4

Job Mix*	L11	L21	L31	L41	L51	Max
0	0.0004	0.0004	0.0005	0.0005	0.0005	0.0005
9.34	0.0669	0.0641	0.0087	0.0097	0.0212	0.0669
48.4	0.0418	0.0908	0.1176	0.0096	0.0793	0.1176
94.1	0.0850	0.0135	0.0488	0.1046	0.0673	0.1046
100	0.0476	0.5636	0.0367	0.0298	0.2697	0.5636

* Percent of all work requests that are type 2 work requests
Note: all values are in seconds

SECTION 8

EVALUATION OF THE CONTROL MODELS

The control models are evaluated on the basis of both the quantitative simulation results discussed in the previous two chapters and various qualitative features which are discussed below.

8.1 Qualitative Aspects of the Models

The qualitative aspects that are investigated include the ability to provide fault-tolerant operation (e.g., graceful degradation and restoration), the ability for the system to expand gracefully, and the ability to balance the system load.

8.1.1 XFDPS.1

The XFDPS.1 model is a truly distributed and decentralized model of control. In this model resources are partitioned along node boundaries and managed by components residing on the same node as the resource. This design enables the system to remain in operation in the presence of a failure. In such a situation, those nodes not available are simply not contacted when queries concerning resources are made. The failed nodes are also not considered as locations for the execution of tasks during the formulation of the work distribution and resource allocation decision.

This model of control requires some activity on the part of all nodes in order to satisfy each work request. There is no single node that is by design supposed to receive any more activity than any other node; instead, the work is spread evenly across all nodes. In addition, global information for the work distribution and resource allocation decision is obtained for each work request as it is processed. This global data enables the control to better balance the load across the network.

This control model is not without its problems. The global searches for resources that occur for every work request may be unnecessary (e.g., in those instances in which only local resources are required). Short local jobs therefore suffer to the advantage of the longer jobs utilizing non-local resources.

8.1.2 XFDPS.2

XFDPS.2 utilizes a single centralized file system directory. On the surface this model appears to be simple to implement. A central directory is maintained, and all file system queries are sent to the node housing that directory. However, problems result when fault-tolerant operation is desired. No longer can a single central directory be maintained because the loss of the node housing the directory would be catastrophic. Alternative strategies which provide for fault-tolerant operation (see for example Garcia-Molina's technique described in [Garc79] for providing fault tolerance in a centralized locking distributed data base system) significantly complicate the design of the control and, while not requiring a large amount of additional effort in order to maintain the information needed to recover from a failure, will require a significant expenditure of resources in order to perform the actual recovery operation. It should be noted that the simulation of XFDPS.2 does not account for the overhead required to provide fault-tolerant operation. Therefore, the average work request response times observed in the experiments may possibly be lower than if the necessary control features for providing fault-tolerant operation were present.

Model XFDPS.2 also presents problems with growth. When a new node is introduced into the system, a large amount of work is required to update the central directory in order to add information about the resources of a new node. This factor can be quantified and will be the subject of future experiments.

8.1.3 XFDPS.3

The XFDPS.3 model is similar to XFDPS.1. It differs in its policy for obtaining file availability information. A local search is made first. If all resources required are found, they are utilized; otherwise, a global search for resources is conducted. As described in Section 5, this model provides faster response to work requests utilizing only local resources, as should be expected. Due to its information gathering policy, the potential for utilizing distant resources in order to balance the load is sacrificed because resource availability on other nodes may never be considered.

8.1.4 XFDPS.4

XFDPS.4 utilizes redundant copies of the file system directory on all nodes. Access to the directory is restricted to the node possessing the control vector that is passed among the nodes of the network. This model tends to work somewhat like a batch system by delaying file system requests until the control vector (CV) is received and then processing these requests as a batch.

The presence of the replicated file directory implies that there is both duplication of information storage and duplication of effort as consistency is maintained across the replicated copies. Since file system requests are delayed until the CV arrives, jobs with very short service times may experience unusually large response times. Finally, as with XFDPS.2, the introduction of a new node requires a large amount of work in order to update the replicated directories.

8.1.5 XFDPS.5

XFDPS.5 is nearly identical to XFDPS.1, differing only in its policy of not locking or in any way reserving resources prior to the formulation of a work distribution and resource allocation decision. With this policy, resources are not expected to be needlessly tied up in most cases. A problem does exist if the chosen resources cannot be locked once selected for allocation. In this case a new resource allocation decision must be made and previously allocated and locked resources may need to be released.

8.1.6 XFDPS.6

XFDPS.6 differs from XFDPS.1 in the manner in which the task graph and task activation are handled. In this model the tasks of a work request that are chosen to execute on the same node are presented to the PROCESS MANAGER of the selected node collectively. A task graph identifying this collection of tasks is constructed and task activation and termination are handled by the PROCESS MANAGER. Thus, the TASK SET MANAGER need send only one message to each of the nodes utilized by the work request in order to activate all tasks. In addition, only one termination message is received from each node. Further savings are provided because the PROCESS MANAGER on the node where the tasks are executing can immediately release the resources utilized by the tasks as each task terminates.

8.2 Quantitative Aspects of the Models

In the previous two chapters it was demonstrated that only work requests with short service times are effected by the type of executive control strategy employed. When the population of work requests consists of jobs requiring little service time, XFDPS.2 provides the best performance followed by XFDPS.3, XFDPS.1, and XFDPS.4 in that order. The performance of XFDPS.4 is noticeably poorer than the other models at all bandwidths.

It is also important to notice that the demands on the communication system by the executive control are not very great. Tables 23 and 24 show the average link queue waiting times for group 1 simulations with XFDPS.1 and XFDPS.2 respectively. Since only control message traffic is present on the communication system in the first group of experiments, the values for average link queue waiting times obtained in those experiments represent the load placed on the communication system by the executive control. The values in Tables 23 and 24 are small indicating that there is very little delay in obtaining access to a communication link. This data demonstrates that the communication systems utilized in these experiments can easily handle the message traffic required to conduct the activities of the executive control.

8.3 Comparison of the Models

On the basis of performance considerations alone, XFDPS.2 is favored over the other control models. It is the consideration of fault-tolerance issues that reduces the desirability of XFDPS.2. Existing strategies for providing a fault-tolerant central directory based system (see [Garc79, Mena78]) require only a small amount of additional work in order to maintain the data structures so that the central directory can be reconstructed if lost due to a failure. One disadvantage of the control strategy of XFDPS.2 is the computation required to reconstruct the central directory when it has been lost. This requires that new work requests not be processed while the data structure is being restored. All work on the system is temporarily delayed for the duration of the reconfiguration process which can conceivably involve a large amount of time. Thus, the operation of all nodes of the system is severely impacted by the loss of the central node.

In contrast, the strategies of XFDPS.1, XFDPS.3, XFDPS.5, and XFDPS.6 provide fault-tolerant operation without unusual delays effecting all nodes as a result of losing a particular node. The resources in these models are partitioned with separate managers for each partition. If a node is lost, the other nodes will simply bypass the manager of the lost node in their search for resources. These models, therefore, seem to be a better choice than XFDPS.2 when considering the objective of enhancing the fault-tolerance of the system.

XFDPS.1, XFDPS.3, XFDPS.5, and XFDPS.6 differ only in certain aspects of control. From the performance data obtained in the third group of experiments, XFDPS.3 must be favored over XFDPS.1. The strategy of not looking for resources on a global basis if they can be found locally prevents XFDPS.3 from optimizing the utilization of system resources (e.g., load balancing). The advantages and disadvantages of XFDPS.5 and XFDPS.6 were discussed earlier.

This analysis demonstrates that the choice of control strategy is not a simple one. It is very dependent on the ultimate goals of the system and the nature of the jobs to be processed on the system. For example, if most of the work involves only local processing, XFDPS.3 is the obvious choice for the system. If on the other hand the distributed facilities are utilized extensively, this model is not necessarily the clear choice. XFDPS.2 may be a better selection if delays due to failures will be tolerated as long as good performance during normal operation is provided. If delays due to failures cannot be tolerated, one of the other control models may be appropriate or a hybrid system utilizing the ideas from XFDPS.3 and XFDPS.6 may be a preferable alternative.

This analysis does not consider model XFDPS.4 when discussing which system should be used under specific or different circumstances. This is due both to the performance problems of this model as demonstrated by the third group of experiments and the fact that it does not have qualitative features that make it more attractive than the any of the other models.

SECTION 9

CONCLUSIONS

This dissertation has discussed the problems of providing an executive control for a Fully Distributed Processing System. The fundamental characteristics of such a control are discussed and several models of control are described. These models are analyzed on the basis of performance data obtained through simulation and the various qualitative features that are identified.

The simulation experiments not only provide data upon which the performance of the various control models can be compared, but they also provide insight into how a Fully Distributed Processing System can perform. It can be concluded that, for the type of jobs processed in the first group of experiments, there is no measurable loss of performance as a result of providing a fully distributed control environment. The average response times computed for work requests on a non-distributed single node network are found to be similar to those for a five node FDPS.

It can also be concluded from the simulation experiments that the message traffic resulting from the operations of the executive control do not present a work load that cannot be easily handled by the communication system. This is indicated by the relatively small magnitude of the average waiting times for the various link queues found in the first group of experiments in which only control message traffic is present on the communication system.

Both the third and fourth groups of experiments demonstrate that not all jobs are insensitive to the control model being utilized. In these experiments jobs with short service times are found to be sensitive to delays attributable to control overhead. With the results from the fourth group of experiments, one can observe the increase in control overhead as the communication system becomes saturated.

The performance data obtained from the third group of experiments indicates that model XFDPS.2, which utilizes a central directory, provides better performance than the other models in a fault-free environment. It is speculated that its performance in the presence of failures, especially a failure involving the node containing the central directory, will be extremely

low. Thus, the other models are favored when fault-tolerance issues are considered.

Future research in this area should concentrate on the issues of fault-tolerance. Specifically, the question of the cost of maintaining the data structures in order to provide a fault-tolerant operation must be addressed. In addition, the impact on the system that is required to recover from a fault must be addressed. It is necessary that the result of these investigations provide quantitative data that can aid system designers in determining the appropriate control strategy for their system.

APPENDIX 1

CONTROL MODEL PSEUDO CODE

1.1 Psuedo Code for the XFDPS.1 Control Model

1.1.1 System initiator

```
 1:  process system_initiator;
 2:  { Every node possesses one of these processes.  This process
 3:    initiates a node in the network by assigning 'task_set_manager'
 4:    processes to each connected user terminal, activating the
 5:    'file_system_manager' process, and activating the
 6:    'processor_utilization_manager' process. }
 7:
 8:  begin
 9:     for every attached user terminal i do
10:        task_set_manager (TERMINAL, i);
11:     endfor;
12:     file_system_manager;
13:     processor_utilization_manager;
14:  end system_initiator;
```

1.1.2 Task Set Manager

```
 1:  process task_set_manager (case input_origin: inp_orig of
 2:                                TERMINAL: (term: terminal_address);
 3:                                CMNDFILE: (fd: filedescriptor)
 4:                             end);
 5:  { Every terminal and every executing command file are assigned
 6:    a 'task_set_manager' process.  When a process of this type
 7:    is activated, one of two sets of parameters is passed to it
 8:    depending upon the source of input to the process.  If the
 9:    process is assigned to handle input from a terminal, the
10:    address of the terminal is provided.  If the process is
11:    assigned to handle input from a command file, the file
12:   descriptor for the command file is provided. }
13:
14:  var
15:     tg: task_graph_pointer;
16:     command_line: string;
17:     msg: message_pointer;
18:
```

```
19:   begin
20:      while <either the terminal is attached or the end
21:         of the file has not been reached> do
22:         <get the next work request and store it in command_line>;
23:         new (tg);
24:         parse (command_line, tg);
25:         <send a message of type M1 (file availability request) to
26:            the file_system_manager on this node that contains the
27:            names of files need for this work request>;
28:         <send a message of type M2 (processor utilization request)
29:            to the processor_utilization_manager on this node>;
30:         <wait for a message from processor_utilization_manager>;
31:         <store processor utilization information in tg^>;
32:         <wait for a message from file_system_manager>;
33:         <store file availability information in tg^>;
34:         if work_distributor_and_resource_allocator (tg) = ERR then
35:            { work distribution and resource allocation
36:            decision could not be made }
37:            <report error>;
38:            if input_origin = CMNDFILE then
39:               exit { leave the loop }
40:            else
41:               next { next iteration of loop }
42:            endif;
43:         endif;
44:         <send a message of type M3 (file lock and release request)
45:            to the file_system_manager on this node>;
46:         <wait for a message from file_system_manager>;
47:         if <all locks could not be applied> then
48:            <report error>;
49:            <send a message of type M4 (file release request)
50:               to the file_system_manager on this node>;
51:            if input_origin = CMNNDFILE then
52:               exit { leave the loop }
53:            else
54:               next { next iteration of loop }
55:            endif;
56:         endif;
57:         for <all files chosen to be copied before execution> do
58:            <send a message of type M5 (file copy request) to the
59:               file_system_manager on this node>;
60:         if <files neeed copying> then
61:            <wait for a message from the file_system_manager>;
62:         endif;
63:         for <each node i chosen to execute parts of the
64:            work request> do
65:            <send a message of type M6 (process activation request)
66:               to the process_manager on node i>;
67:         endfor;
```

```
68:          repeat
69:             <wait for a termination message from a process_manager
70:                or a request to terminate the command file from
71:                the process_manager that activated this
72:                task_set_manager>;
73:             if <this is a termination message from a
74:                process_manager> then
75:                <mark the terminated task as completed in tg^>;
76:                <send a message of type M4 (file release request)
77:                   to the file_system_manager on this node>;
78:                if <the termination status indicated that the
79:                   process terminated due to an error> then
80:                   for <each node i still running parts of this
81:                      work request> do
82:                      <send a message of type M7 (process kill request)
83:                         to the process_manager on node i>;
84:                   endfor;
85:                endif;
86:             else
87:                for <every task of the work request> do
88:                   if <the task has not completed> then
89:                      <send a message of type M7 (process kill request)
90:                         to the process_manager responsible for
91:                         the task>;
92:                   endif;
93:                endfor;
94:                break; { exit the loop }
95:             endif;
96:          until <all tasks have terminated>;
97:       endwhile;
98:    end task_set_manager;
```

1.1.3 File System Manager

```
 1:    process file_system_manager;
 2:    { Every node possesses one of these processes.  This process
 3:      satisfies various requests concerning the file system.
 4:      This is accomplished by communicating with the
 5:      file_set_managers on all nodes. }
 6:
 7:    var
 8:       msg: message_pointer;
 9:       favptr: file_availability_rec_pointer;
10:       flrprt: file_lock_and_release_rec_pointer;
11:
12:    begin
13:       loop
14:          <wait for a message of any type (let msg point to
15:             the message)>;
```

```
16:          case msg^.message_type of
17:             M1: { file availability information request }
18:                begin
19:                   new (favptr);
20:                   <insert the record favptr points to into the
21:                      list of fav_recs>;
22:                   <record names of files identified in msg^>;
23:                   for <each node i> do
24:                      <send a message of type M8 (file availability
25:                         request) to the file_set_manager on node i
26:                         that contains the names of all files>;
27:                   endfor;
28:                end;
29:             M3: { file lock and release request }
30:                begin
31:                   new (flrptr);
32:                   <insert the record flrptr points to into the
33:                      list of flr_recs>;
34:                   for <each node i> do
35:                      <send a message of type M9 (file lock and
36:                         release request) to the file_set_manager
37:                         on node i that contains the names of all
38:                         files from msg^ that are identified
39:                         as being located at node i>;
40:                   endfor;
41:                end;
42:             M4: { file release request }
43:                begin
44:                   for <each node i> do
45:                      <send a message of type M10 (file release
46:                         request) to the file_set_manager on
47:                         node i that contains the names of all
48:                         files from msg^ that are identified as
49:                         being located at node i>;
50:                   endfor;
51:                end;
52:             M5: { file copy request }
53:                begin
54:                   new (fmvptr);
55:                   <insert the record fmvptr points to into the list
56:                      of fmv_recs>;
57:                   for <each file named in msg^> do
58:                      <insert the file name into fmvptr^>;
59:                      <send a message of type M11 (create file
60:                         request) to the file_set_manager on the node
61:                         where the file is to be copied>;
62:                   endfor;
63:                end;
```

```
64:            M12: { file availability info from file_set_manager }
65:               begin
66:                  <let favptr point to the fav_rec that msg^
67:                     is a response to>;
68:                  <fill in the availability information in favptr^>;
69:                  if <responses from all file_set_managers
70:                     have been received> then
71:                     <send a message of type M16 (file availability
72:                        information) to the task_set_manager
73:                        identified by a field of favptr^>;
74:                  endif;
75:               end;
76:            M13: { lock and release results from file_set_manager }
77:               begin
78:                  <let flrptr point to the flr_rec that msg^
79:                     is a response to>;
80:                  <fill in the lock and release results in flrptr^>;
81:                  if <responses from all file_set_managers
82:                     that were contacted have been received> then
83:                     <send a message of type M17 (results of file
84:                        lock & release request) to task_set_manager
85:                        identified by a field of flrptr^>;
86:                  endif;
87:               end;
88:            M14:{result of file creation req. from file_set_manager}
89:               begin
90:                  { This message is part of a series of messages
91:                    used to copy a file from one node to another.
92:                    At this point, file processes have been
93:                    activated at both the sending and receiving
94:                    nodes.  The next step is to send a signal to
95:                    the sending process to begin transmission. }
96:                  <send a message of type M18 (signal to begin copy)
97:                     to the sending file process in the copy
98:                     operation>;
99:               end;
100:           M15: { copy completion signal from a file process }
101:              begin
102:                 <let fmvptr point to the fmv_rec that msg^
103:                    is a response to>;
104:                 <record in fmvptr^ that the copy operation
105:                    indicated in msg^ has been completed>;
106:                 if <all copy operations have been completed> then
107:                    <send a message of type M19 (results of file
108:                       copy request) to the task_set_manager
109:                       identified by a field of fmvptr^>;
110:                 endif;
111:              end;
112:        endcase;
113:     endloop;
114:  end file_system_manager;
```

1.1.4 Processor Utilization Manager

```
 1:  process processor_utilization_manager;
 2:  { Every node possesses one of these processes.  This process
 3:    records the latest processor utilization information received
 4:    from each node's processor_utilization_monitor; it provides
 5:    task_set_managers with this information on demand; and if it
 6:    does not hear from a processor_utilization_monitor within a
 7:    particular interval of time, it records the processor as down
 8:    and attempts to contact that processor_utilization_monitor. }
 9:
10:  var
11:     msg: message_pointer;
12:     pcutil: array [NODES_OF_THE_NET] of pc_utilization;
13:
14:  begin
15:     loop
16:        <wait for a message of any type (let msg point to
17:           the message)>;
18:        case msg^.message_type of
19:           M2: { pc utilization information request }
20:              begin
21:                 <send a message of type M20 (pc utilization
22:                    information) to the task_set_manager that
23:                    sent the message and is identified in msg^>;
24:              end;
25:           M3: { pc utilization information from monitor }
26:              begin
27:                 <record information in msg^ in pcutil [msg^.node]>;
28:                 <reset deadman timer for information arriving
29:                    from node msg^.node>;
30:              end;
31:           M22: { deadman timer signal - this indicates that a
32:                  processor_utilization_monitor has not reported
33:                  within the required time }
34:              begin
35:                 pcutil [msg^.node] := NOT_AVAILABLE;
36:                 <send a message of type M23 ("are you alive?"
37:                    query) to the processor_utilization_monitor
38:                    on node msg^.node>;
39:              end;
40:        endcase;
41:     endloop;
42:  end processor_utilization_manager;
```

1.1.5 Processor Utilization Monitor

```
 1:  process processor_utilization_monitor;
 2:  { Every node possesses one of these processes.  This process
 3:    records various performance measurements and computes a
 4:    processor utilization value that is periodically transmitted
 5:    to all processor_utilization_managers. }
 6:
 7:  begin
 8:     loop
 9:        <gather performance measurements>;
10:        <compute processor utilization value>;
11:        for <each node i> do
12:           <send a message of type M21 (processor utilization
13:              information) to the processor_utilization_manager
14:              on node i>;
15:        endfor;
16:        <sleep until it is time to gather more measurements>;
17:        <wait until it is time to gather more measurements
18:           or a message from a processor_utilization_manager
19:           arrives>;
20:     endloop;
21:  end processor_utilization_monitor;
```

1.1.6 Process Manager

```
 1:  process process_manager;
 2:  { Every node possesses one of these processes.  This process
 3:    manages the processes that are executing on its node. }
 4:
 5:  var
 6:     pcbptr: process_control_block_pointer;
 7:     process_name_table: process_name_to_pcbptr_map;
 8:     msg: message_pointer;
 9:
10:  begin
11:     loop
12:        <wait for the arrival of a message (let msg point
13:           to the message)>;
14:        case msg^.message_type of
15:           M6: { process activation request }
16:              begin
17:                 if <process type is an object file> then
18:                  new (pcbptr);
19:                  <record process identifying information
20:                     and pcbptr in process_name_table>;
21:                  <fill in the necessary information in pcbptr^>;
22:                  <initiate the loading of the process>;
23:                 else
24:                  task_set_manager (CMNDFILE, msg^.file_descriptor);
25:                  <record process identifying information
26:                     and task_set_manager identification in
27:                     process_name_table>;
```

```
28:                  endif;
29:               end;
30:            M7: { process kill request }
31:               begin
32:                  <find the process in process_name_table>;
33:                  if <the process is an object file> then
34:                     <terminate the process>;
35:                     <unload the process>;
36:                     <dispose of the process control block>;
37:                     <send a message of type M24 (process
38:                        termination message) to the task_set_manager
39:                        that activated the process>;
40:                  else { the process is a command file }
41:                     <send message of type M25 (request to terminate
42:                       the execution of a command file) to the
43:                       task_set_manager executing this command file>;
44:                  endif;
45:               end;
46:         endcase;
47:      endloop;
48:   end process_manager;
```

1.1.7 File Set Manager

```
 1:   process file_set_manager;
 2:   { Every node possesses one of these processes.  This process
 3:     manages the files located on its node. }
 4:
 5:   var
 6:      msg: message_pointer;
 7:      file_directory: file_location_information;
 8:
 9:   begin
10:      loop
11:         <wait for the arrival of a message (let msg point
12:            to the message)>;
```

```
13:          case msg^.message_type of
14:             M8: { file availability request }
15:                begin
16:                   for <each file named in msg^> do
17:                      <search for the file>;
18:                      if <the file was found> then
19:                         if <the file is free> then
20:                            <reserve the file>;
21:                            <record the desired access to the file>;
22:                            <note that the file is available>;
23:                         else
24:                            if <the desired access to the file
25:                               is READ> and <the access already
26:                               granted to the file is READ> then
27:                               <note that the file is available>;
28:                            else
29:                               <note that the file is not available>;
30:                            endif;
31:                         endif;
32:                      else
33:                         <note that the file is not available>;
34:                      endif;
35:                   endfor;
36:                   <send a message of type M12 (file availability
37:                      information) to the file_system_manager
38:                      on node msg^.node>;
39:                end;
40:             M9: { file lock and release request }
41:                begin
42:                   for <each file in msg^> do
43:                      <search for the file>;
44:                      if <the file was found> then
45:                         <lock or release the file as requested>;
46:                      else
47:                         <note that request could not be satisfied>;
48:                      endif;
49:                   endfor;
50:                   <send a message of type M13 (results of file lock
51:                      and release request) to the file_system_manager
52:                      on node msg^.node>;
53:                end;
54:             M10: { file release request }
55:                begin
56:                   for <each file in msg^> do
57:                      <search for file and release the lock on it>;
58:                   endfor;
59:                end;
```

```
60:            M11: { file creation request }
61:               begin
62:                  <create an entry for a new file in file_directory>;
63:                  <activate a file process for the file>;
64:                  <send a message of type M14 (results of file
65:                     creation) to the file_system_manager on
66:                     node msg^.node>;
67:               end;
68:         endcase;
69:      endloop;
70:   end file_set_manager;
```

1.2 Psuedo Code for the XFDPS.2 Control Model

1.2.1 System Initiator

Same as XFDPS.1.

1.2.2 Task Set Manager

XFDPS.1 with the following changes:

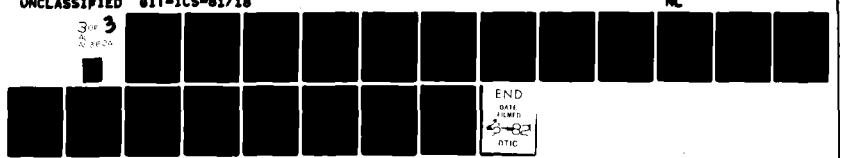
```
25:   <send a message of type M2 (file availability request) to
26:      the file_system_manager on node 1 that contains the
27:      names of files needed for this work request>;

44:   <send a message of type M3 (file lock and release request)
45:      to the file_system_manager on node 1>;

76:   <send a message of type M4 (file release request)
77:      to the file_system_manager on node 1>;
```

1.2.3 File System Manager

(complete replacement)

```
 1:   process file_system_manager;
 2:   { This process resides on node 1 and satisfies various requests
 3:     concerning the file system.  This process maintains the
 4:     centralized file system directory. }
 5:
 6:   var
 7:      msg: message_pointer;
 8:
 9:   begin
10:      loop
11:         <wait for a message of any type (let msg point to
12:            the message)>;
13:         case msg^.message_type of
14:            M1: { file availability information request }
15:               begin
16:                  for <each file named in msg^> do
17:                     <search for the file>;
18:                     if <the file was found> then
19:                        for <each node i> do
20:                           if <the file is free on node i> then
21:                              <reserve the file>;
22:                              <record desired access to the file>;
23:                              <note that the file is available on
24:                                 node i>;
```

AD-A113 624 GEORGIA INST OF TECH ATLANTA SCHOOL OF INFORMATION A--ETC F/G 9/2
DISTRIBUTED AND DECENTRALIZED CONTROL IN FULLY DISTRIBUTED PROC--ETC(U)
DEC 81 T G SAPONAS N00014-79-C-0873
UNCLASSIFIED GIT-ICS-81/18 NL



```
25:                         else
26:                            if <the desired access to the file
27:                               is READ> and <the access already
28:                               granted to the file is READ> then
29:                               <note that the file is available on
30:                                  node i>;
31:                            else
32:                               <note that the file is not available
33:                                  on node i>;
34:                            endif;
35:                         endif;
36:                      endfor;
37:                   else
38:                      <note that the file is not available on
39:                         any node>;
40:                   endif;
41:                endfor;
42:                <send a message of type M12 (file availability
43:                   information) to the task_set_manager requesting
44:                   the information>;
45:             end;
46:          M3: { file lock and release request }
47:             begin
48:                for <each file in msg^> do
49:                   <search for the file>;
50:                   if <the file was found and is present
51:                      on the node specified> then
52:                      <lock or release the file as requested>;
53:                   else
54:                      <note that request could not be satisfied>;
55:                   endif;
56:                endfor;
57:                <send a message of type M13 (results of file lock
58:                   and release request) to the task_set_manager
59:                   that made the request>;
60:             end;
61:          M4: { file release request }
62:             begin
63:                for <each file in msg^> do
64:                   <search for file and release the lock on it>;
65:                endfor;
66:             end;
67:       endcase;
68:    endloop;
69: end file_system_manager;
```

1.2.4 Process Utilization Manager

Same as XFDPS.1.

1.2.5 Processor Utilization Monitor

Same as XFDPS.1.

1.2.6 Process Manager

Same as XFDPS.1.

1.3 Pseudo Code for the XFDPS.3 Control Model

1.3.1 System Initiator

Same as XFDPS.1.

1.3.2 Task Set Manager

Same as XFDPS.1.

1.3.3 File System Manager

XFDPS.1 with the following changes:

```
23:   <send a message of type M8 (file availability
24:      request) to the file_set_manager on the same node
25:      as this file_system_manager>;
26:
27:

69:   if <this response is from this node> and
70:      <all files have not been found available> then
71:      for <every other node i> do
72:         <send a message of type M8 (file availability
73:            request) to the file_set_manager on node i>;
74:      endfor;
74a:  else
74b:     if <responses from all file_set_managers have been
74c:        received or all files have been found locally> then
74d:        <send a message of type M16 (file availability
74e:           information) to the task_set_manager identified
74f:           by a field of favptr^>;
74g:     endif;
74h:  endif;
```

1.3.4 Process Utilization Manager

Same as XFDPS.1.

1.3.5 Processor Utilization Monitor

Same as XFDPS.1.

1.3.6 Process Manager

Same as XFDPS.1.

1.6.7 File Set Manager

Same as XFDPS.1.

1.4 Pseudo Code for the XFDPS.4 Control Model

1.4.1 System Initiator

Same as XFDPS.1.

1.4.2 Task Set Manager

Same as XFDPS.1.

1.4.3 File System Manager

```
process file_system_manager;
{ Every node possesses one of these processes.  This process
  satisfies various requests concerning the file system and
  helps maintain the redundant copies of the file system
  directory. }

var
   msg: message_pointer;

begin
   loop
      <wait for a message of any type (let msg point to
         the message)>;
      case msg^.message_type of
         M1, M3, M4: { availability, lock, and release requests }
            begin
               <place the message on the queue of file system
                  requests arriving at this node>;
            end;
         CV: { control vector }
            begin
               while <the file system request queue is
                  not empty> do
                  <remove a message from the queue (let msg point
                     to the message)>;
                  case msg^.message_type of
                     M1: { file availability information request }
                        begin
                           for <each file named in msg^> do
                              <search for the file>;
                              if <the file was found> then
                                 for <each node i> do
                                    if <file free on node i> then
                                       <reserve the file>;
                                       <record the desired access
                                          to the file>;
                                       <note that the file is
                                          available on node i>;
```

```
39:                                    else
40:                                       if <desired access to file
41:                                          is READ> and <access
42:                                          already granted is READ>
43:                                          then
44:                                          <note that the file is
45:                                             available on node i>;
46:                                       else
47:                                          <note that file is not
48:                                             available on node i>;
49:                                       endif;
50:                                    endif;
51:                                 endfor;
52:                              else
53:                                 <note that the file is not
54:                                    available on any node>;
55:                              endif;
56:                           endfor;
57:                           <send a message of type M12 (file
58:                            availability information) to the
59:                            task_set_manager requesting the info>;
60:                     end;
61:                  M3: { file lock and release request }
62:                     begin
63:                        for <each file in msg^> do
64:                           <search for the file>;
65:                           if <the file was found & is present
66:                              on the node specified> then
67:                              <lock or release the file>;
68:                           else
69:                              <note failure to satisfy request>;
70:                           endif;
71:                        endfor;
72:                        <send a message of type M13 (results of
73:                           file lock and release request) to
74:                           task_set_manager that made request>;
75:                     end;
76:                  M4: { file release request }
77:                     begin
78:                        for <each file in msg^> do
79:                           <search for file and release lock>;
80:                        endfor;
81:                     end;
82:               endcase;
83:            endwhile;
84:            <send a message of type UPV (update vector) to the
85:               next node (according to the predetermined
86:               ordering of nodes) containing the changes just
87:               made to the file system directory>;
88:      end;
```

```
 89:           UPV: { update vector }
 90:              begin
 91:                 if <this UPV created by this node> then
 92:                    <send a message of type CV (control vector) to
 93:                       the next node (according to the predetermined
 94:                       ordering of nodes)>;
 95:                 else
 96:                    <update the file system directory>;
 97:                    <send the message of type UPV (update vector)
 98:                       to the next node (according to the
 99:                       predetermined ordering of nodes)>;
100:                 endif;
101:              end;
102:        endcase;
103:     endloop;
104:  end file_system_manager;
```

1.4.4 Process Utilization Manager

Same as XFDPS.1.

1.4.5 Processor Utilization Monitor

Same as XFDPS.1.

1.4.6 Process Manager

Same as XFDPS.1.

1.5 Pseudo Code for the XFDPS.5 Control Model

1.5.1 System Initiator

Same as XFDPS.1.

1.5.2 Task Set Manager

Same as XFDPS.1.

1.5.3 File System Manager

Same as XFDPS.1.

1.5.4 Process Utilization Manager

Same as XFDPS.1.

1.5.5 Processor Utilization Monitor

Same as XFDPS.1.

1.5.6 Process Manager

Same as XFDPS.1.

1.5.7 File Set Manager

XFDPS.1 with the following changes:

```
20:   <note that the file is available>;
21:
22:
```

1.6 Pseudo Code for the XFDPS.6 Control Model

1.6.1 System Initiator

Same as XFDPS.1.

1.6.2 Task Set Manager

XFDPS.1 with the following changes:

```
75:   for <each task in the message> do
76:      <mark the task as completed in tg^>;
77:   endfor;

87:   for <every node i still executing parts of the work
88:      request> do
89:      <send a message of type M7 (process kill request)
90:         to the process_manager on node i>;
91:   endfor;
92:
93:
```

1.6.3 File System Manager

Same as XFDPS.1.

1.6.4 Process Utilization Manager

Same as XFDPS.1.

1.6.5 Processor Utilization Monitor

Same as XFDPS.1.

1.6.6 Process Manager

```
 1:   process process_manager;
 2:   { Every node possesses one of these processes.  This process
 3:     manages the processes that are executing on its node. }
 4:
 5:   var
 6:      pcbptr: process_control_block_pointer;
 7:      process_name_table: process_name_to_pcbptr_map;
 8:      subtg: task_graph_pointer;
 9:      msg: message_pointer;
10:
```

```
11:  begin
12:     loop
13:        <wait for the arrival of a message (let msg point
14:           to the message)>;
15:        case msg^.message_type of
16:           M6: { process activation request }
17:              begin
18:                 new (subtg);
19:                 for <each task i im msg^> do
20:                  <record task i in subtg^>;
21:                  if <task i names an object file> then
22:                   new (pcbptr);
23:                   <record process identifying information
24:                      and pcbptr in process_name_table>;
25:                   <fill in necessary information in pcbptr^>;
26:                   <initiate the loading of the process>;
27:                  else
28:                   task_set_manager (CMNDFILE, msg^.file_descriptor);
29:                   <record process identifying information
30:                      and task_set_manager identification in
31:                      process_name_table>;
32:                  endif;
33:                 endfor;
34:                 <link subtg^ onto the list of subtaskgraphs
35:                    executing on this node>;
36:              end;
37:           M7: { process kill request }
38:              begin
39:                 <find the subtaskgraph in the list of
40:                    subtaskgraphs executing on this node (let
41:                    subtg point to the subtaskgraph)>;
42:                 for <each task i in subtg^> do
43:                    if <task i has not completed> then
44:                       if <task i names an object file> then
45:                          <terminate the process>;
46:                          <unload the process>;
47:                          <dispose of the process control block>;
48:                          <mark task i as terminated>;
49:                       else { the process is a command file }
50:                          <send a message of type M25 (request to
51:                           terminate execution of command file) to
52:                           task_set_manager running this cmnd file>;
53:                       endif;
54:                    endif;
55:                 endfor;
```

```
56:                  if <all the tasks in subtg^ have completed> then
57:                     <send a message of type M24 (subtaskgraph
58:                        termination message) to the task_set_manager
59:                        that activated the subtaskgraph>;
60:                     <remove subtg^ from the list of subgraphs
61:                        executing on this node>;
62:                     dispose (subtg);
63:                  endif;
64:               end;
65:         endcase;
66:      endloop;
67:   end process_manager;
```

BIBLIOGRAPHY

[Abra80] Abraham, Steven M., and Dalal, Yogen K., "Techniques for Decentralized Management of Distributed Systems," COMPCON Spring 80 (February 25-28, 1980): 430-437.

[Abra70] Abramson, N., "The Aloha System - Another Alternative for Computer Communications," AFIPS Conference Proceedings 37 (1970 Fall Joint Computer Conference): 281-285.

[Aiso75] Aiso, H., Matsushita, Y., et. al., "A Minicomputer Complex - KOCOS," IEEE/ACM Fourth Data Communications Symposium (October, 1975): 5-7 to 5-12.

[Akin80] Akin, T. Allen, Flinn, Perry B., and Forsyth, Daniel H., "User's Guide for the Software Tools Subsystem Command Interpreter (The Shell)," School of Information and Computer Science, Georgia Institute of Technology, 1980.

[Alsb78] Alsberg, P. A., Belford, G. G., Day, J. D., and Grapa, E., "Multi-Copy Resiliency Techniques," COMPSAC 78 Tutorial: Distributed Data Base Management (November 13-16, 1978): 128-176.

[Ande75] Anderson, George A., and Jensen, E. Douglas., "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples," Computing Surveys 4 (December, 1975): 197-213.

[Andr81] Andre, Jean-Pierre, and Petit, Jean-Claude, "GALAXIE: A Reconfigurable Network of Processors with Distributed Control," Proceedings of the Second International Conference on Distributed Computing Systems (April, 1981): 86-94.

[Arno80] Arnold, R. G., Ramseyer, R. R., Wind, L. B., and Householder, E. A., "MMBC Architecture," Proceedings of the First International Conference on Distributed Computing Systems (October, 1979): 707-724. also submitted for publication in IEEE Transactions on Computers, received March 31, 1980.

[Bach78a] Bachman, Charles W., "Domestic and International Standards Activities for Distributed Systems," COMPCON Fall 78 (September, 1978): 140-143.

[Bach78b] Bachman, Charles, and Canepa, Mike, "The Session Control Layer of an Open System Interconnection," COMPCON Fall 78 (September, 1978): 150-156.

[Ball76] Ball, J. Eugene, Feldman, Jerome, Low, James R., Rashid, Richard, and Rovner, Paul, "RIG, Rochester's Intelligent Gateway: System Overview," IEEE Transactions on Software Engineering SE-2 (December, 1976): 321-328.

[Balz71] Balzer, R. M., "PORTS - A Method for Dynamic Interprogram Communication and Job Control," AFIPS Conference Proceedings 38 (1971 Spring Joint Computer Conference): 485-489.

[Bart78] Bartlett, J. F., "A 'Nonstop' Operating System," Proceedings of the Hawaii International Conference on System Sciences vol. III (January, 1978): 103-117.

Bibliography

[Bask77] Baskett, F., Howard, J., and Montague, J., "Task ... DEMOS," Proceedings of the Sixth ACM Symposium on Operating ... Principles (November, 1977): 23-31.

[Boeb78] Boebert, W. E., Franta, W. R., Jensen, E. D., an ... "Decentralized Executive Control in Distributed Com... COMPSAC 78 (November, 1978): 254-258.

[Boeb78] Boebert, W. E., Franta, W. R., Jensen, E. D., and Kai... nel Primitives of the HXDP Executive," COMPSAC 78 ... 595-600.

[Brin78] Brinch Hansen, Per, "Distributed Processes: A Concurre... Concept," Communications of the ACM 21 (November, 19...

[Brya81] Bryant, R. M., and Finkel, R. A., "A Stable Distribute... Algorithm," Proceedings of the Second International ... Distributed Computing Systems (April, 1981): 314-323.

[Caba79a] Cabanel, J. P., Marouane, M. N., Besbes, R., Sazbon, ... Diarra, A. K., "A Decentralized OS Model for ARAMIS Distr... puter System," Proceedings of the First International ... Distributed Computing Systems (October, 1979): 529-53...

[Caba79b] Cabanel, J. P., Sazbon, R. D., Diarra, A. K., Marouane, ... Besbes, R., "A Decentralized Control Method in a ... System," Proceedings of the First International ... Distributed Computing Systems (October, 1979): 651-65...

[Carr70] Carr, C. S., Crocker, S. D., and Cerf, V. G., "Host/Host ... tion Protocol in the ARPA Network," AFIPS Conference Proceed... (1970 Spring Joint Computer Conference): 589-597.

[Case77] Casey, L., and Shelness, N., "A Domain Structure for ... Computer Systems," Proceedings of Sixth ACM Symposium on ... Systems Principles (November, 1977): 101-108.

[Chu80] Chu, Wesley W., Holloway, Leslie J., Lan, Min-Tsung, and ... "Task Allocation in Distributed Data Processing," ... (November, 1980): 57-69.

[Clar80] Clark, David D., and Svobodova, Liba, "Design of Distribute... Supporting Local Autonomy," COMPCON Spring 80 (February, ... 444.

[Cook80] Cook, R. P., "*MOD - A Language for Distributed Programm... Transactions on Software Engineering SE-6 (November, 198...

[Cott78] Cotton, Ira W., "Computer Network Interconnection," Computer Networks 2 (1978): 25-34.

[Denn78] Denning, Peter J., "Operating Systems Principles for Data Flow Networks," Computer (July, 1978): 86-96.

[desJ78] desJardins, Richard, and White, George, "ANSI Reference Model for Distributed Systems," COMPCON Fall 78 (September, 1978): 144-149.

[Dion80] Dion, Jeremy, "The Cambridge File Server," Operating Systems Review 14 (October, 1980): 26-35.

[DuBo81] DuBois, D., "Distributed Systems Simulator for Computer Networks," General Electric Technical Information Series No. 80C1S004.

[Eckh78] Eckhouse, Richard H., Jr., Stankovic, John A., and van Dam, Andries, "Issues in Distributed Processing - An Overview of Two Workshops," Computer (January, 1978): 22-26.

[Ensl78] Enslow, Philip H., Jr., "What is a 'Distributed' Data Processing System?" Computer (January, 1978): 13-21.

[Ensl81a] Enslow, Philip H., Jr., and Saponas, Timothy G., "Distributed and Decentralized Control in Fully Distributed Processing Systems - A Survey of Applicable Models," Technical Report No. GIT-ICS-81/02, Georgia Institute of Technology, February, 1981.

[Ensl81b] Enslow, Philip H., Jr., and Saponas, Timothy G., "Distributed and Decentralized Control in Fully Distributed Processing Systems - Evaluation of Models," Technical Report No. GIT-ICS-81/09, Georgia Institute of Technology, July, 1981.

[Falk78] Falk, Gilbert, "A Comparison of Network Architectures - the ARPANET and SNA," Proceedings of the National Computer Conference (1978): 755-763.

[Farb72a] Farber, David J., and Kenneth C. Larson, "The System Architecture of the Distributed Computer System - The Communications System," Proceedings of the Symposium on Computer-Communications Networks and Teletraffic (April, 1972): 21-27.

[Farb72b] Farber, D. J., and Larson, K. C., "The Structure of the Distributed Computing System - Software," Proceedings of the Symposium on Computer-Communications Networks and Teletraffic (April, 1972): 539-545.

[Feld79] Feldman, J. A., "High Level Programming for Distributed Computing," Communications of the ACM 22 (June, 1979): 353-368.

[Feld78] Feldman, J. A., Low, J. R., and Rovner, P. D., "Programming Distributed Systems," Proceedings of the 1978 Annual Conference (December, 1978): 310-316.

[Flei81] Fleisch, Brett D., "An Architecture for Pup Services on a Distributed Operating System," Operating Systems Review 15 (January, 1981): 26-44.

[Flet80] Fletcher, John G., and Watson, Richard W., "Service Support in a Network Operating System," COMPCON Spring 80 (February 25-28, 1980): 415-424.

[Fors77] Forsdick, Harry C., Schantz, Richard E., and Thomas, Robert H., "Operating Systems for Computer Networks," Bolt Beranek and Newman Technical Report 3614 (July, 1977).

[Fors78] Forsdick, Harry C., Schantz, Richard E., and Thomas, Robert H., "Operating Systems for Computer Networks," Computer (January, 1978): 48-57.

[Fras75] Fraser, A. G., "A Virtual Channel Network," Datamation 21 (February, 1975): 51-56.

[Fusi81] Fusi, A., and Sommi, G., "Distributed Virtual Systems," Proceedings of the Second International Conference on Distributed Computing Systems (April, 1981): 41-49.

[Garc79] Garcia-Molina, H., "Performance Comparison of Update Algorithms for Distributed Databases, Crash Recovery in the Centralized Locking Algorithm," Progress Report No. 7, Stanford University, 1979.

[Garn80] Garnett, N. H., and Needham, R. M., "An Asynchronous Garbage Collector for the Cambridge File Server," Operating Systems Review 14 (October, 1980): 36-40.

[Gord79] Gordon, R. L., Farr, W. W., and Levine, P., "Ringnet: A Packet Switched Local Network with Decentralized Control," Proceedings of the Fourth Conference on Local Computer Networks (October, 1979).

[Gord78] Gordon, Robert L., and Test, Jack A., "PR1ME IPC Conference Report," PR1ME Research Note 107, October, 1978.

[Gray77] Gray, J. P., "Network Services in Systems Network Architecture," IEEE Transactions on Communications COM-25 (January, 1977): 104-116.

[Hoar74] Hoare, C. A. R., "Monitors: An Operating System Structuring Concept," Communications of the ACM 17 (October, 1974): 549-557.

[Hoar78] Hoare, C. A. R., "Communicating Sequential Processes," Communications of the ACM 21 (August, 1978): 666-677.

[Hopp79] Hopper, K., Kugler, H. J., and Unger, C., "Abstract Machines Modelling Network Control Systems," Operating Systems Review 13 (January, 1979): 10-24.

[Huen77] Huen, Wing, Greene, Peter, Hochsprung, Ronald, and El-Dessouki, Ossama, "TECHNEC, A Network Computer for Distributed Task Control," Proceedings of the 1st Annual Rocky Mountain Symposium on Microcomputers: Systems, Software, Architecture (August 31 - September 2, 1977): 161-194.

[Jens78] Jensen, E. Douglas, "The Honeywell Experimental Distributed Processor - An Overview," Computer (January, 1978): 28-38.

[Jens81] Jensen, E. Douglas, "Distributed Computer Systems," Computer Science Research Review 1979-80, Department of Computer Science, Carnegie-Mellon University, pp. 53-63.

[Jone76] Jones, Anita K., Chansler, Robert J., Jr., Durham, Ivor, Feiler, Peter, and Schwans, Karsten, "Software Management of Cm* - A Distributed Multiprocessor," AFIPS Conference Proceedings 46 (1977 National Computer Conference): 657-663.

[Jone79a] Jones, Anita K., Chansler, Robert J., Jr., Durham, Ivor, Schwans, Karsten, and Vegdahl, Steven R., "StarOS, A Multiprocessor Operating System for the Support of Task Forces," Proceedings of the Seventh Symposium on Operating Systems Principles (December, 1979): 117-127.

[Jone79b] Jones, A., and Schwans, K., "TASK Forces: Distributed Software for Solving Problems of Substantial Size," Proceedings of the Fourth International Conference on Software Engineering (September, 1979): 315-330.

[Jone80] Jones, Anita K., and Schwarz, Peter, "Experience Using Multiprocessor Systems - A Status Report," Computing Surveys 12 (June, 1980): 121-165.

[Keme78] Kemen, H., and Nagel, H. H., "Experiences with a Virtual Network Machine Concept for an Inhomogeneous Local Computer Network," COMPCON Fall 78 (September, 1978): 280-286.

[Kieb81] Kieburtz, Richard G., "A Distributed Operating System for the Stony Brook Multicomputer," Proceedings of the Second International Conference on Distributed Computing Systems (April, 1981): 67-79.

[Kiel79] Kiely, S. C., "An Operating System for Distributed Processing - DPPX," IBM Systems Journal 18 (1979): 507-525.

[Kimb76] Kimbleton, Stephen R., and Mandell, Richard L., "A Perspective on Network Operating Systems," AFIPS Conference Proceedings 45 (1976 National Computer Conference): 551-559.

[Kimb78] Kimbleton, Stephen R., Wood, Helen M., and Fitzgerald, M. L., "Network Operating Systems - An Implementation Approach," Proceedings of the National Computer Conference (1978): 773-782.

[Klei68] Kleinrock, Leonard, "Certain Analytic Results for Time-Shared Processors," Information Processing 68, North-Holland Publishing Co., Amsterdam, 1968, pp. 838-845.

[Lars79] Larson, Robert E., Tutorial: Distributed Control, presented at the First International Conference on Distributed Computing Systems (October 1-4, 1979).

[LeBl81] LeBlanc, Richard J., and Maccabe, Arthur B., "PRONET: Language Features for Distributed Programming," Technical Report No. GIT-ICS-81/03, Georgia Institute of Technology, May, 1981.

[LeLa77] Le Lann, G., "Distributed Computing - Towards a Formal Approach," Information Processing 77 (August 8-12, 1977): 155-160.

[LeLa79] Le Lann, G., "An Analysis of Different Approaches to Distributed Computing," Proceedings of the First International Conference on Distributed Computing Systems (October, 1979): 222-232.

[LeLa81] Le Lann, Gerard, "A Distributed System for Real-Time Transaction Processing," Computer 14 (Febrary, 1981): 43-48.

[Lein58] Leiner, A. L., Notz, W. A., Smith, J. L., and Weinberger, A., "PILOT, the NBS Multicomputer System," Proceedings of the Eastern Joint Computer Conference (December, 1958): 71-75.

[Lisk79] Liskov, B., "Primitives for Distributed Computing," Proceedings of the 7th ACM Symposium on Operating Systems Principles (December, 1979): 33-42.

[Live79] Livesey, N. J., "Inter-process Communication and Naming in the Mininet System," COMPCON Spring 79 Proceedings (February, 1979).

[Live80] Livesey, N. J., "Run-Time Control in a Transaction-Oriented Operating System," Ph.D. Thesis, University of Waterloo, 1980.

[Live78a] Livesey, N. J., and Manning, Eric, "Protection in a Transaction Processing System," Proceedings of the Seventh Texas Conference on Computing Systems (October, 1978).

[Live78b] Livesey, Jon, and Manning, Eric, "What MININET Has Taught Us About Programming Style," COMPSAC 78 (November, 1978): 692-697.

[Lori79] Lorin, H., "Distributed Processing: An Assessment," IBM Systems Journal 18 (1979): 582-603.

[Lu78] Lu, Priscilla M., "A System for Resource-Sharing in a Distributed Environment - RIDE," COMPSAC 78 (November, 1978): 427-433.

[Lunn81] Lunn, K., and Bennett, K. H., "An Algorithm for Resource Location in a Loosely Linked Distributed Computer System," Operating Systems Review 15 (April, 1981): 16-20.

[Mann76] Manning, Eric G., Howard, R., O'Donnel, C. G., Pammett, K., and Chang, E., "A UNIX-based Local Processor and Network Access Machine," Computer Networks 1 (1976): 139-142.

[Mann77] Manning, Eric G., and Peebles, Richard W., "A Homogeneous Network for Data-sharing Communications," Computer Networks 1 (1977): 211-224.

[McQu78] McQuillan, John M., "Message Addressing Modes for Computer Networks," COMPCON Fall 78 (September, 1978): 80-88.

[McQu77] McQuillan, John M., and Walden, David C., "The ARPA Network Design Decisions," Computer Networks 1 (1977): 243-289.

[Mena78] Menasce, Daniel A., Popek, Gerald J., and Muntz, Richard R., "Centralized and Hierarchical Locking in Distributed Databases," COMPSAC 78 Tutorial: Distributed Data Base Management (November, 1978): 178-195.

[Metc76] Metcalfe, R. M., and Boggs, D. R., "Ethernet - Distributed Packet Switching for Local Computer Networks," Communications of the ACM 19 (July, 1976): 395-404.

[Mile78] Milenkovic, Milan, "Relocation and Accessing of Shared Modules in Distributed Processing Systems," Technical Report University of Massachusetts, November, 1978.

[Mill81] Miller, Barton, and Presotto, David, "XOS: An Operating System for the X-Tree Arcitechture," Operating Systems Review 15 (April, 1981): 21-32.

[Mill76] Mills, David L., "An Overview of the Distributed Computer Network," AFIPS Conference Proceedings 45 (1976 National Computer Conference): 523-531.

[Mill77] Millstein, R. E., "The National Software Works: A Distributed Processing System," Proceedings of the ACM Annual Conference (October 16-19, 1977): 44-52.

[Mitr79] Mitrani, I., and Sevcik, K. C., "Evaluating the Trade-off Between Centralized and Distributed Computing," Proceedings of the First International Conference on Distributed Computing Systems (October 1-3, 1979): 520-528.

[Morg77] Morgan, Howard L., and Levin, K. Dan, "Optimal Program and Data Locations in Computer Networks," Communications of the ACM 20 (May, 1977): 315-322.

[Mott79] Mott, Donald R., "A Distributed Computing Architecture for Real-Time System Control and Information Processing," Proceedings of the First International Conference on Distributed Computing Systems (October 1-3, 1979): 204-211.

[Muke79] Mukerji, Jishnu, and Kieburtz, Richard B., "A Distributed File System for a Hierarchical Multicomputer," Proceedings of the First International Conference on Distributed Computing Systems (October, 1979): 448-458.

[Nels80] Nelson, David L., "Application Engineering on a Distributed Computer Architecture," COMPCON Spring 80 (February, 1980): 425-429.

[Oust80a] Ousterhout, John K., Scelza, Donald A., and Sindhu, Pradeep S., "Medusa: An Experiment in Distributed Operating System Structure," Communications of the ACM 23 (February, 1980): 92-105.

[Oust80b] Ousterhout, John K., "Partitioning and Cooperation in a Distributed Multiprocessor Operating System: Medusa," Ph.D. Thesis, Department of Computer Science, Carnegie-Mellon University, April, 1980.

[Peeb78] Peebles, Richard, and Manning, Eric, "System Architecture for Distributed Data Management," Computer (January, 1978): 40-47.

[Peeb80] Peebles, Richard, and Dopirak, Thomas, "ADAPT: A Guest System," COMPCON Spring 80 (February, 1980): 445-454.

[Pete79] Peterson, James L., "Notes on a Workshop on Distributed Computing," Operating Systems Review 13 (July, 1979): 18-30.

[Rowe73] Rowe, Lawrence A., Hopwood, Marsha D., and Farber, David J., "Software Methods for Achieving Fail-Soft Behavior in the Distributed Computing System," 1973 IEEE Symposium on Computer Software Reliability (April, 1973): 7-11.

[Sapo80] Saponas, Timothy G., and Crews, Phillip L., "A Model for Decentralized Control in a Fully Distributed Processing System," COMPCON Fall 80 (September, 1980).

[Sche78] Scherr, A. L., "Distributed Data Processing," IBM Systems Journal 17 (1978): 324-343.

[Shar81] Sharp, Donald D., Jr., "Work Distribution in a Fully Distributed Processing System," Ph.D. Thesis, School of Information and Computer Science, Georgia Institute of Technology, December, 1981.

[Shoc78] Shoch, John F., "Inter-Network Naming, Addressing, and Routing," COMPCON Fall 78 (September, 1978): 72-79.

[Sinc80] Sincoskie, W. David, and Farber, David J., "SODS/OS: A Distributed Operating System for the IBM Series/1," Operating Systems Review 14 (July, 1980): 46-54.

[Smit79] Smith, Reid G., "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver," Proceedings of the First International Conference on Distributed Computing Systems (October, 1979): 185-192.

[Smit80] Smith, Reid G., "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver," IEEE Transactions on Computers C-29 (December, 1980): 1104-1113.

[Solo79] Solomon, Marvin H., and Finkel, Raphael A., "The Roscoe Distributed Operating System," Proceedings of the Seventh Symposium on Operating Systems Principles (December, 1979): 108-114.

[Souz81] Souza, Robert J., and Balkovich, Edward E., "Impact of Hardware Interconnection Structures on the Performance of Decentralized Software," Proceedings of the 8th Annual Symposium on Computer Architecture (May, 1981): 357-365.

[Srin80a] Srini, Vason P., "Framework for Communication in Loosely Coupled Multiple Processor Systems," Proceedings of the 1980 Parallel Processing Conference (,1980):

[Srin80b] Srini, Vason P., and Shriver, Bruce D., "Abstract Dataflow Protocol for Communication in Distributed Computer Systems," COMPCON Fall 80 (September, 1980): 321-330.

[Ston78] Stone, Harold S., and Bokhari, Shahid H., "Control of Distributed Processes," Computer (July, 1978): 97-106.

[Suns77] Sunshine, Carl, "Interprocess Communication Extensions for the UNIX Operating System: I. Design Considerations," Rand Technical Report R-2064/1-AF, June 1977.

[Svob79] Svobodova, Liba, Liskov, Barbara, and Clark, David, "Distributed Computer Systems: Structure and Semantics," MIT Technical Report MIT/LCS/TR-215 (March, 1979).

[Swan76a] Swan, Richard J., Bechtolsheim, Andy, Lai, Kwok-woon, and Ousterhout, John K., "The Implementation of the Cm* Multi-Microprocessor," AFIPS Conference Proceedings 46 (1977 National Computer Conference): 645-655.

[Swan76b] Swan, R. J., Fuller, S. H., and Siewiorek, D. P., "Cm* - A Modular, Multi-Microprocessor," AFIPS Conference Proceedings 46 (1977 National Computer Conference): 637-644.

[Tane81] Tanenbaum, Andrew S., "An Overview of the Amoeba Distributed Operating System," Operating Systems Review 15 (July, 1981): 51-64.

[Thom78] Thomas, Robert H., Schantz, Richard E., and Forsdick, Harry C., "Network Operating Systems," Bolt Beranek and Newman Report No. 3796 (March, 1978).

[Thur78] Thurber, Kenneth J., "Computer Communication Techniques," COMPSAC78 Proceedings (November, 1978): 589-594.

[vanT81] van Tilborg, Andre M., and Wittie, Larry D., "Wave Scheduling: Distributed Allocation of Task Forces in Network Computers," Proceedings of the Second International Conference on Distributed Computing Systems (April, 1981): 337-347.

[Ward80] Ward, Stephen A., "TRIX: A Network-Oriented Operating System," COMPCON Spring 80 (February, 1980): 344-349.

[Wilk80] Wilkes, Maurice V., and Needham, Roger M., "The Cambridge Model Distributed System," Operating Systems Review 14 (January, 1980): 21-29.

[Witt79] Wittie, Larry D., "A Distributed Operating System for a Reconfigurable Network Computer," Proceedings of the First International Conference on Distributed Computing Systems (October, 1979): 669-677.

[Witt81] Wittie, Larry D., "Communication Structures for Large Networks of Microcomputers," IEEE Transactions on Compputers C-30 (April, 1981): 264-273.

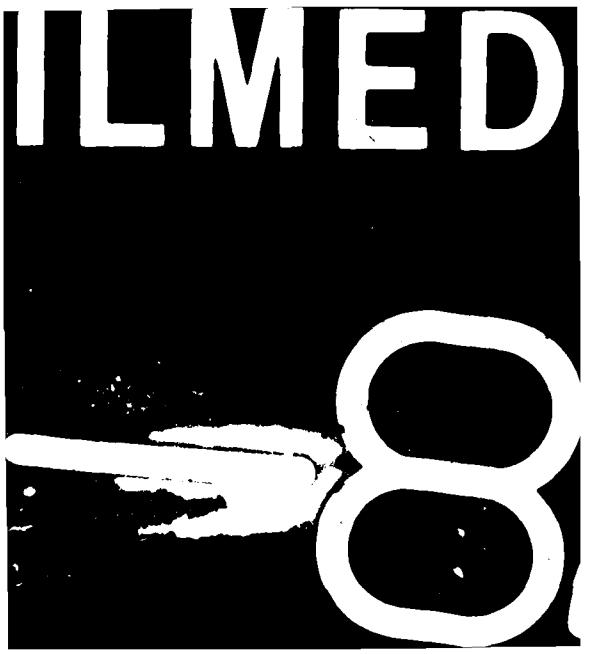
[Witt80] Wittie, Larry D., and Van Tilborg, Andre M., "MICROS, A Distributed Operating System for MICRONET, A Reconfigurable Network Computer," IEEE Transactions on Computers C-29 (December, 1980): 1133-1144.

[Wulf72] Wulf, William A., and Bell, C. G., "C.mmp - A Multi-Mini-Processor," AFIPS Conference Proceedings 41 part II (1972 Fall Joint Computer Conference): 765-777.

[Wulf81] Wulf, William A., Levin, Roy, and Harbison, Samuel P., HYDRA/C.mmp An Experimental Computer System, McGraw-Hill Book Company, New York, 1981.

[Zimm80] Zimmerman, H., "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection," IEEE Transactions on Communications COM-28 (April, 1980): 425-432.

[Zuck77] Zucker, Steven, "Interprocess Communication Extensions for the UNIX Operating System: II. Implementation," Rand Technical Report R-2064/2-AF, June, 1977.


ILMED
8